NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AUV DIVE CONTROL SYSTEM DEVELOPMENT
INCLUDING SENSOR BIAS COMPENSATION
AND PARAMTER ESTIMATION

by

Gerard J. Reina

December 1988

Thesis Advisor: A.J. Healey

ABSTRACT

The U.S. Navy and a number of its contractors are presently developing unmanned miniature submarines for several vital underwater missions. These include surveillance, submarine tracking, and bottom mapping. Foregoing Reserarch at NPS produced a "testbed" as a research platform for demonstrating the performance of AUVs. Combining the power of an IBM PC/AT in conjunction with a high level programming language, a state space dive control system was developed and instituted for the 30 inch AUV model. Parameter Estimation using a Recursive Least Squares Fit scheme and a State Observer were incorporated in the controller. Procedures dealing with hardware/software interfacing, AUV simulation analysis, and computation speed of large programming code were investigated.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

I am extremely grateful to my thesis advisor, the Chairman of the Mechanical Engineering Department, Dr. Anthony J. Healey for his endless patience and emphatic support. His enthusiasm was the driving force behind this research.

I also wish to thank CDR Gordon MacDonald for his unselfish support and cooperation throughout the vehicle testing phase.

I. INTRODUCTION

A. GENERAL

The crux of this thesis is to develop, test and evaluate a real-time state-space derived dive plane controller for the NPS prototype Autonomous Underwater Vehicle (AUV). The U.S. Navy and a number of its contractors are presently developing unmanned miniature submarines (AUV's) for several important underwater missions. These include surveillance, submarine tracking, submarine decoys, bottom surveying, and off-board sensor support for both surface ships and submarines [Ref. 1]. The shape of these vehicles is similar to that of a large torpedo and can be deployed on submarines, and surface ships.

The AUV testbed at the Naval Postgraduate School (NPS) currently being developed is to examine the use of, and problems associated with, advanced control technologies. The objectives include:

- determine requirements imposed by the implied need for artificial intelligence,
- development of advanced computer control concepts and architecture,
- determine system interface feasibility. [Ref. 2]

Unlike the conventional body-of-revolution design, the NPS testbed is similar to a low aspect ratio wing. It is similar conceptually to the late model Swimmer Delivery

Vehicle (SDV) design, but actually based on a two and three to one aspect ratio box shape vehicle. Vehicle selection was based initially on the availability and thoroughness to which the vehicle hydrodynamic characteristics were modeled and validated, providing for ease of program development and verification [Ref. 3].

Modern trends in control system design are moving towards greater complexity. The desire for complex tasks and greater accuracy fuel the need for high powered computers to accomplish real-time control tasks in short duration. One of the main interests here is to determine the capabilities of real-time control with adaptive autopilots and various computer architectures. The need for adaptivity lies in the need to achieve robust performance over a wide range of operating conditions. Also, although computational speed may suffer, rapid response is seen to be important when evasive and/or obstacle avoidance maneuvers are called for at the supervisory/expert system level. It follows that the design trade-off between response and robustness requires examination. For these reasons, this particular work has concentrated on the development of a real-time controller including the ability to perform parameter estimation, and the issues surrounding its response time performance.

B. REVIEW OF PREVIOUS WORK

The initial proposal from the Naval Postgraduate School for research on an Autonomous Underwater Vehicle began 1 October 1987. It was titled "Navigation, Path Planning, Dynamics and Control of Generic Autonomous Underwater Vehicles," and was submitted to the Naval Surface Weapons Center in Silver Spring, Maryland. The primary stated objective was to assess the capabilities for the design, construction, testing, and operation of a testbed as a research platform for demonstrating the performance of AUVs [Ref. 2].

The first task involved selecting a suitable submersible model. A non-conventional wing-shaped model with body cross sections nearly rectangular rather than circular became the selected choice.

Using Dynamic Simulation Language (DSL), Boncal (1987) modified the existing equations of motion (DTNSRDC 2510) and performed all the necessary differentials for their linearization to suit the needs of the AUV controller development. The linearization was necessary to the design of model-based controls and to simulate controlled responses for all six degrees of freedom of vehicle motion [Ref. 3]. Both linear and non-linear models were developed and evaluated. This lead to the initial development of a controller robust enough to handle a variety of reflexive type maneuvers over a wide range of speeds (in the programming language DSL).

In order to investigate the real-time control issues, and to provide a means to generate experimental data for the evaluation of parameter identification methods, a later effort by Brunner (1988), set out to design, build, and test a model 30 inches in length and 7 inches wide. The model was self-propelled and remotely controlled via radio transmission. It was equipped with rate gyro sensors, a pressure cell depth sensor and a pressure cell speed sensor. Two DC motors provided the propulsion power [Ref. 4]. Open-loop dive plane dynamic response tests were performed in a water tank where response measurements of depth were made by pressure cells, and pitchrate by a pitchrate gyro. Data from each run were recorded using PCLAB and the DT NOTEBOOK [Ref. 5] data acquisition software, and analyzed using the MATRIXx [Ref. 6] data analysis capability. Comparing these data with the computer model simulation, hydrodynamic coefficients and system transfer functions were developed.

Initial developments of a digital autopilot to support the NPS test vehicle were made by Delaplane (1988), wherein an IPM PC/AT was used in conjunction with DATA TRANSLATION Analog to Digital (A/D) interfacing boards, as the platform for the real-time closed loop controller. A simplified control program for AUV dive plane commands had been implemented. Signal generators were used to replicate AUV transducer outputs of depth, speed, and pitchrate to verify system operation [Ref. 7]. The inclusion of real control laws, and the system parameter identification, however, was left to the activity reported herein.

C. SCOPE OF WORK

The scope of this investigation was to develop the digital autopilot [Ref. 7] into a three state feedback, Single Input multiple Output, real-time closed-loop dive control system for the NPS AUV model. Initial stages of research lead to the notion that the pitch-heave coupling prevalent in many underwater vehicles was not significant for this vehicle. The three state variables of the dynamic model were then reduced to the pitch rate, pitch angle, and the water depth. Because of tank length limitations it was foreseen that an analog computer model of the vehicle dynamics would be essential to the ease of the real time control development, and therefore, a scaled simulator was built.

The final stages of program development were to implement the digital dive plane controller compatible with the 30 inch AUV model. Considering the small vehicle size, and relatively high component costs, the NPS AUV was unable to include a pitch angle sensor. Since pitch angle feedback was later shown to be critical to the autopilot stability, where the pitch angle could not be measured directly, a state reconstruction system was needed and had to be designed. This led to the development of a full state observer driven by measured inputs and outputs [Ref. 8]. The Observer, including the identification of pitch rate sensor bias, the bias compensation, and its real time performance was large part of this effort.

D. APPROACH

Primarily, a solid understanding of the software/ hardware used was essential. In this control system, a high level programming language (Turbo Pascal) was interfaced with an A/D translation board (hardware) via a software package (PCLAB). References 9, 10, 11 established a sufficient base on which the elaborate program schemes could be developed effectively. Once accomplished, the challenge of debugging large programs could readily be resolved.

Secondly, incorporating experimental data from previous AUV research proved paramount. This research thesis could not be accomplished without the foundation set by preceding studies.

Lastly, using real-time feedback of AUV model dynamics via the analog computer strengthened the ease by which program development could proceed. The Analog Computer simulated the model dynamics including the sensor bias and, partially, the effects of changing speed. This provided for the preliminary design of a robust state-space controller.

E. WHAT FOLLOWS

In Chapter II, some detail is provided concerning the models of the vehicle dynamics. Chapter III discusses the analog computer modelling. Chapter IV gives an account of the program development. Chapters V and VI contain a discussion of results and Chapter VII is a summary and conclusion.

II. SIMPLIFIED MODELING OF VEHICLE DIVE DYNAMICS

A. GENERAL

In this chapter, a review of earlier experiments conducted under the work of Brunner [Ref. 4], appears and the details of the third-order model of the vehicle is given. This leads to a discussion of the design issues in the controller and the observer systems and the special problem of identifying the pitch rate sensor bias from input and output data.

B. REVIEW OF OPEN LOOP EXPERIMENTS

Open loop experiments were conducted during January, 1988 (Brunner), in which the model AUV was driven by open loop command to perform dive plane maneuvers. From these test runs, MATRIXx datafiles were constructed for both the command voltage to the dive plane and the measured voltages corresponding to the pitch rate, and the water depth responses. Using the linear model time history response features in MATRIXx, together with many simulation runs in which hydrodynamic parameters were varied both in value and in combinations of values, several "best fit" parameter combinations were synthesized to represent the dynamics of the model AUV. It was discovered that the influence of coupling between the heave mode, and the pitch mode dynamics was not strong in determining the pitch rate behavior, and that for the purposes of control shape design, the heave motion would be neglected. The influence of the power cord was found to be a determining factor in the overall heave response, but future tests would be conducted by neutralizing the weight of the cord, and this effect would not adversely affect closed loop results.

The numerical values of the vehicle parameters, with their respective dimensional units, were established as follows in the next section.

C. THIRD-ORDER MODEL FOR DIVE-PLANE DYNAMICS

The vehicle equations of motion in the dive plane are the surge, heave, and pitch motion equations together with the kinematical relationships linking the global longitudinal and vertical velocity components to the body fixed surge, heave, and pitch rates. These are given in detail in Boncal (1987). When linearized about a constant speed straight line flight path, the relationships simplify to four first-order state equations in the heave and pitch rates and positions. In short,

$$\dot{W} = \frac{ZW*UX*W}{MZDW} + \frac{ZQ*UX*q}{MZDW} - \frac{ZD*UX^2*\delta}{L*MZDW} + \frac{F1}{MZDW}$$

$$\dot{q} = \frac{MW*UX*W}{L^2*IYY} + \frac{MQ*UX*q}{L*IYY} + \frac{MD*UX^2*\delta}{L^2*IYY} + \frac{F2}{IYY}$$

$$\dot{\theta} = q$$

$$\dot{z} = W - UX*\theta$$

where w, q, θ, z are the four state variables and the parameters are given by,

HEAVE DAMPING	ZW = -1.5
PITCH CROSS COUPLING	ZQ = 0.0
HEAVE CROSS COUPLING	MW = 0.0
PITCH DAMPING	MQ = -0.15
MOMENT EFFECT	ZD = 0.0
MOMENT EFFECT	MD = 0.225
MASS PLUS ADDED MASS	MZDW = 1.005
INERTIA PLUS ADDED MASS	IYY = 0.072
EFFECT OF TETHER	F1 = 0.11
EFFECT OF TETHER	F2 = 0.0
FORWARD SPEED	UX = 2.1 FEET/SEC
LENGTH	L = 2.5 FEET,

where the units of the parameters are given in Table 2.1.

TABLE 2.1

PARAMETER UNITS

Parameter	Units	Value (Dimensional)	Value (Volts Units)
MQ*UX/L*IYY	sec^{-1}	10.0 1/sec	1.8
UX	ft/sec	10.0 ft/sec	2.1
$MD*UX^2/L^2IYY$	rad/sec^2	$10.0\ 1/sec^2$	2.531

It should be pointed out that the matching of experimental data was done by comparison of sensor output voltage response time history and that proper conversion to dimensional values was essential to being able to translate to control gains for real time control system design.

Because it was found that the effects of the heave/pitch coupling were not strong, and that the influence of w was dominated by the power cord, it was assumed that for the purpose of control system design, the heave dynamics would remain unmodelled, and the model base would be reduced to third-order with the equations as follows:

$$\underline{x}' = [q, \theta, Z],$$

$$\dot{\underline{x}} = \begin{bmatrix} -1.8 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & -2.1 & 0.0 \end{bmatrix} \underline{x} + \begin{bmatrix} 2.531 \\ 0.0 \\ 0.0 \end{bmatrix} \delta$$

D. FULL STATE FEEDBACK CONTROL DESIGN

For the nominal forward speed of 2.1 ft/sec, the system dynamics matrix, and the system input and output matrices are:

$$A = \begin{bmatrix} -1.8 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & -2.1 & 0.0 \end{bmatrix} \qquad B = \begin{bmatrix} 2.531 \\ 0.0 \\ 0.0 \end{bmatrix} \qquad C = [0.0, 0.0, 1.0]$$

These equations are characterized by the open loop eigenvalues:

$$v = [-1.8, 0.0, 0.0]$$

which properly indicates the first-order pole for the pitch rate response and the double integration from pitch rate to depth. The output matrix has diagonal coefficients with either unity values or other constants depending on whether the output signals are considered to be sensor output voltages or the dimensional values of the system state variables.

Noting that the open loop system is completely controllable in the formal sense, a pole placement technique was determined to be appropriate for the autopilot design. Additionally, since the overall system required a digital controller, discrete time methods were needed, and the question of sample rate arose.

Digital systems for the control of continuous processes are limited to providing a sequence of stepwise constant levels that can be updated every sample time, but remain fixed within that interval. Clearly, a short interval is preferred. However, the execution time of extensive amounts of code requires that the sampling period must be sufficient for code execution, but it must not be so short as to places limitations on the closed loop poles as specified for the control design, being consistent with the execution time of the real time code.

In balancing the foregoing, two systems were designed. The first, at 20 Hz, was designed to accommodate a control code that executed a full three state feedback design where the system parameter identification was bypassed. The second was at 5 Hz (the maximum upper limit possible under the present hardware configuration) where parameter identification was performed.

The open loop system dynamics matrix and input matrix were then discretized using the 20 Hz sample rate so that the new A, and B matrices became:

$$A = \begin{bmatrix} .9139 & 0.0 & 0.0 \\ .0478 & 1.0 & 0.0 \\ -.0025 & -.1050 & 1.0 \end{bmatrix} \qquad B = \begin{bmatrix} .121 \\ .0031 \\ -.0001 \end{bmatrix}$$

Closed loop poles for the 20 Hz system were then specified at

$$s = [0.92, 0.925, 0.926]$$

in the z-domain, corresponding to three poles in the s-plane with time constants equal to 1.0 sec, with the following state feedback gains,

$$Kc*x = [0.3066, 0.8604, -0.2493],$$

where allowances were made for the output transducer gain.

For a state regulator system, the control law is,

$$u = -Kc*x$$

when constant non-zero command signals are to be introduced, the control law requires modification to,

$$u = N*r - Kc*x$$

in which the command(s) r and the set point scaling matrix, N, are specified so that the tracking error r-y vanishes. The closed loop system equations become

$$x = (A-B*Kc)*x+B*N*r; \qquad y = C*x$$

and the tracking error vanishes in steady state when,

$$C*(A-B*Kc)^{-1}*B*N=I; \; N=(C*(A-B*Kc)^{-1}*B)$$

yielding the discrete time control law used,

$$u(k) = -0.2493*[r(k)-z(k)]-z(k)]$$
$$-0.8604*[\;(k)]-0.3066*[q(k)]$$

This control provides reasonable response time and calls for a control surface activity level that would exceed the stroke of the vehicle dive plane.

E. OBSERVATION OF UNMEASURED STATES

In the development of the model AUV the size limitation prevented the inclusion of an angle sensor for pitch, so the control system was required to reconstruct that signal in order to yield stable results. The application of a full state observer, configured as a model based compensator is given by the following one step prediction equations,

$$x(k+1) = A*\hat{x}(k) + B*u(k) + Ko*[y(k)-C*\hat{x}(k)]$$

in which y(k) is the previous time sensor output signals. In this case, two outputs were available to the observer, the pitch rate, and the water depth. The observer gains, Ko, may be selected by using the dual of the controller pole placement technique, or by using Linear Quadratic Regulator optimization. Although pole placement techniques in general are based on a single input signal, two inputs can be handled by the assignment of some weighting value (i.e., equal) ineffective towards the outcome. The effect of the two signals, y(k), now becomes, y(k) = [1;1]*ye(k)

so that the gains for the two-input system become

Ko = [1;1]*ke

where Ke are found via pole placement using ye(k).

The inclusion of the observation (and later parameter identification) algorithms required the use of a slower sample rate, 5 Hz. Consequently, enclosed loop poles of the observer were chosen as a compromise between fast response and the resulting sensitivity to extraneous noise when implemented as a model based compensator. The following were the final selection,

Ko = [0.0;-0.4029;0.5677]

F. PARAMETER ESTIMATION OF SENSOR BIAS

Parameter identification of the rate gyro bias was accomplished by the use of an ARMA model of the gyro system itself where the input variable was the dive plane angle and the output was pitch rate. The ideal transfer function between input and output is a first order system which has two numerator coefficients, and two denominator coefficients to identify. (Note that the z-domain transfer function is to be used because we are dealing with sequences of sampled data.) With this form, the parameter vector and the data vector can be written as,

$$\theta = [1\ a1\ b0\ b1]; \quad x' = [y(t)\ y(t-1)\ d(t)\ d(t-1)];$$

and

$$\theta{*}x = \text{bias}(t).$$

The foregoing is then repeated and a least squares fit for θ is then obtained after the bias is included inside the vector θ. Least squares solution of these equations yields both the vector of parameters, θ, and the unknown bias. In this work, a recursive least squares algorithm was used with a Householder routine to compute the recursive gains which is time consuming but robust.

III. ANALOG SIMULATOR

A. GENERAL

Simulation, a must for prototype development, provides a source of numeric, kinematic and logical awareness which is vital in designing a complex vehicle system and analyzing its performance. It provides an environment in which the many possible states of the system, and the transitions between them, can be exercised and observed in an accessible and controlled manner. Properly utilized, simulation will reduce significantly the amount of field testing required. Design and performance deficiencies can be identified early in the development cycle, before the system is deployed [Ref. 12].

A technique known as "Hybrid" simulation employs a combination of analog and digital computing [Ref. 13]. Using the hydrodynamic coefficients of the AUV model, the analog computer can be "programmed" to simulate AUV vehicle dynamics. With the control program running and a "target depth" entered as a keyboard input, AUV pitch angle, depth, and pitchrate, calculated by the analog computer, are fed back to the digital computer. The three states are compared to a model reference and an appropriate dive plane command is determined. The result is a continuous update (real-time control) of vehicle dive plane angle (see Figure 3.1 for hardware configuration).

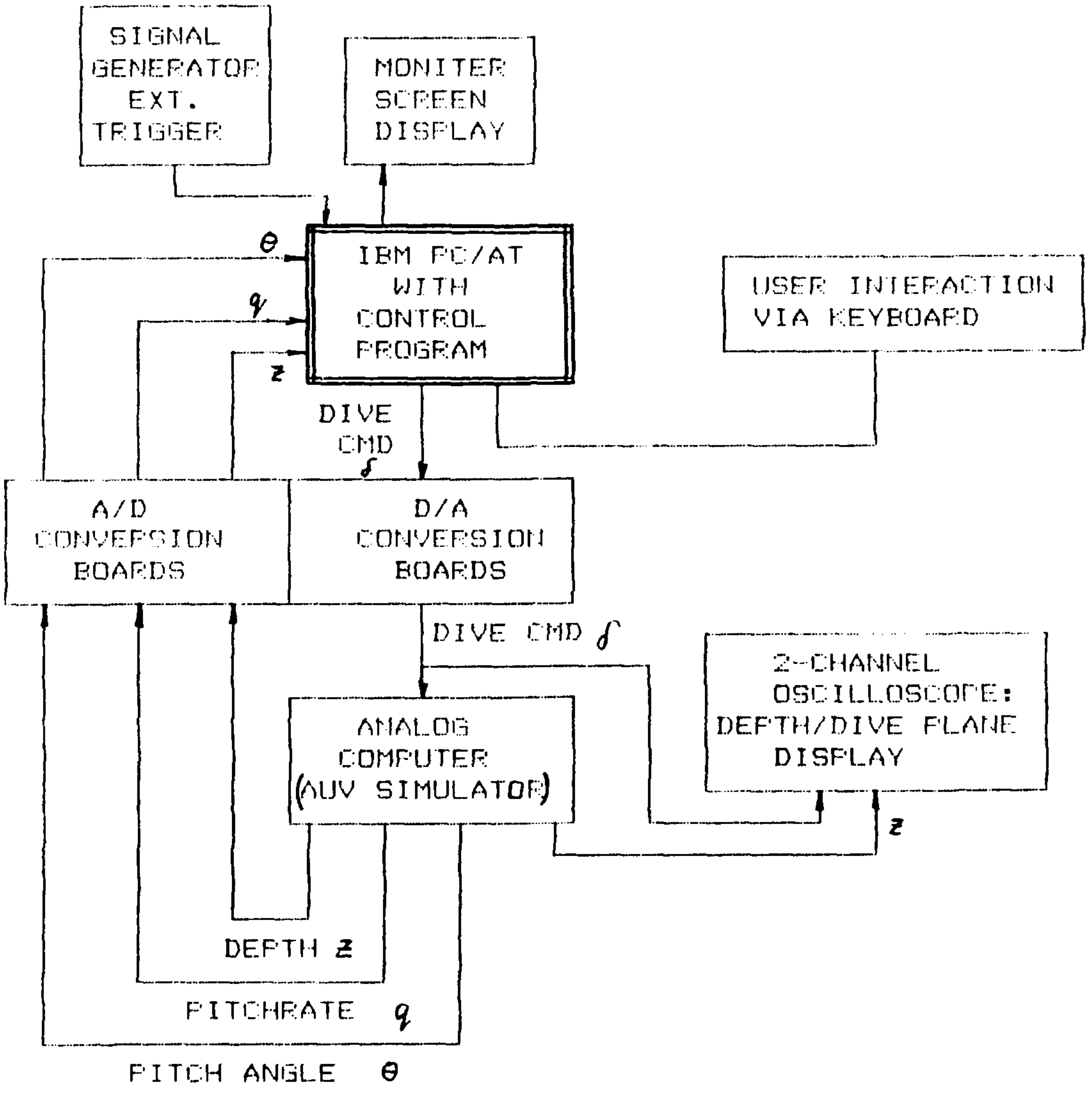
In the second phase of program development, the analog computer was configured to sustain a constant or variable bias on the system. This lead to the development of a bias estimator.

B. DESCRIPTION OF ANALOG SIMULATOR

In developing the analog simulations, the equations of motion of the vehicle are required to be converted. Convenient conversion or scaling is based on the maximum variable values being scaled with a +/- 10 volts maximum output on each amplifier. In analog simulations, variable summations and integrations are performed by summing and integrating operational amplifiers. Multiplication by constant coefficients are performed by attenuating potentiometers set to the appropriate fraction of 1.0. With scaling consistently applied through the 1 or 10:1 input, appropriate gains are sent to the corresponding amplifiers.

Figure 3.2 shows an example of a first order system simulation using an analog operational amplifier. The equation is:

$$\dot{q} = [1.5]\,\delta \text{ rad/sec}$$



SIGNAL GENERATOR EXT. TRIGGER
MONITER SCREEN DISPLAY
IBM PC/AT WITH CONTROL PROGRAM
USER INTERACTION VIA KEYBOARD
θ
q
z
DIVE CMD δ
A/D CONVERSION BOARDS
D/A CONVERSION BOARDS
DIVE CMD δ
2-CHANNEL OSCILLOSCOPE: DEPTH/DIVE PLANE DISPLAY
ANALOG COMPUTER (AUV SIMULATOR)
z
DEPTH z
PITCHRATE q
PITCH ANGLE θ

Figure 3.1 Hardware Configuration

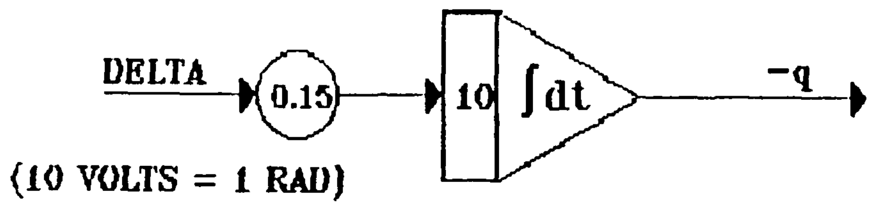


Figure 3.2 Example of a First Order System Using an Analog Operational Amplifier

where δ is in radians. Note the coefficient 1.5 rad/sec per radian is set at 0.15 in the potentiometer with a gain of 10 in the integrating amplifier.

In Figure 3.3, the diagram represents the third order model of the vehicle given by Equations in II.C. Table 2.1 gives a listing of the system parameter values, their dimensions, and the potentiometer settings used in the simulations.

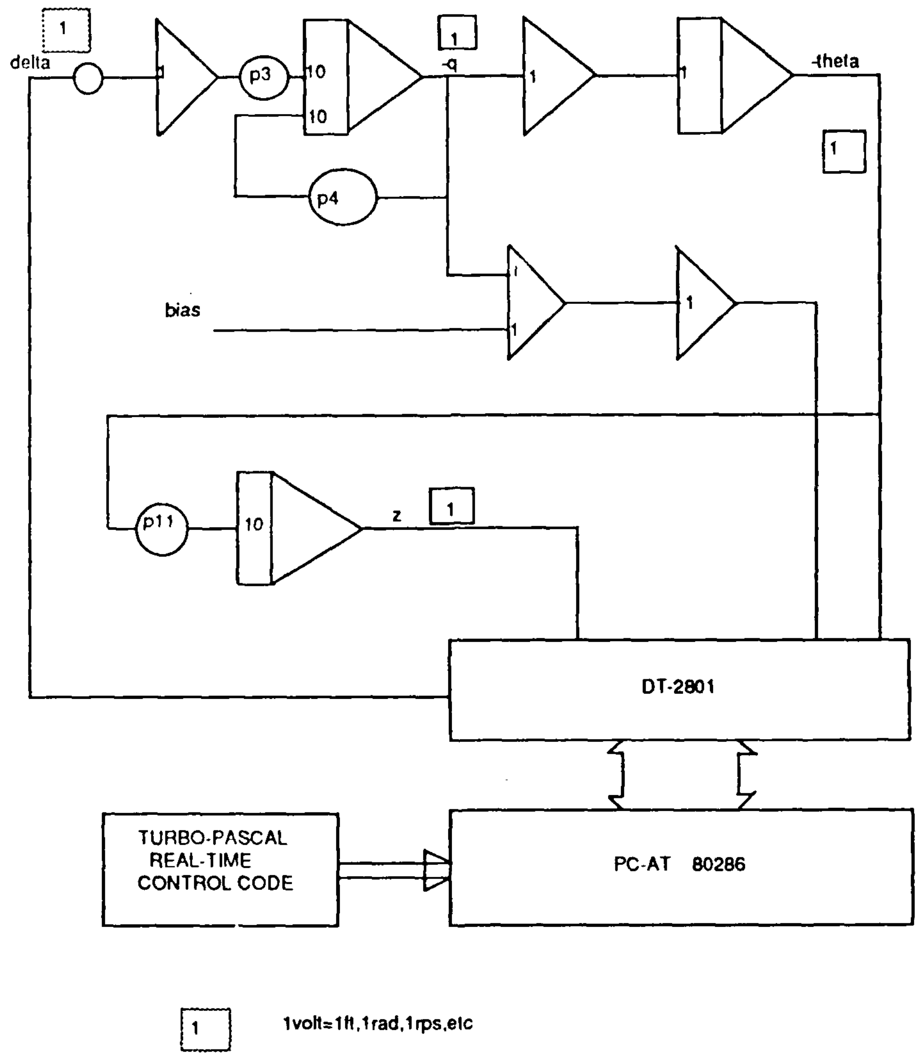


Figure 3.3 Vehicle Simulation Analysis Using an Analog Computer

IV. DIGITAL CONTROL PROGRAM DEVELOPMENT

A. GENERAL

As explained in Reference 7, this digital control program represents a functional component at the lowest level of the digital autopilot echelon. The digital control program interfaces with the servos controlling the AUV dive planes. The "handshaking" of data between the digital controller and the voltage-operated servos onboard the AUV model are accomplished by A/D interface hardware [Ref. 14]. In Figure 3.1 the hardware arrangement using the analog computer as a simulator of sensor feedback is exhibited.

Much of the program development of Reference 7 is implemented into this thesis. A number of dilemmas had to be overcome for program progression to continue.

The preliminary program code had parameter conversions of feet/sec, inches, and degrees/sec. To alleviate valuable computation time, and to better support the AUV model, the code for the input and ouput (I/O) dimensions was modified. All I/O parameter dimensions were converted to voltages.

Significant concern focused on the programmable interrupt control. Previous work was hampered when software interaction was unachievable. The preliminary control program would run only when initial keyboard inputs of target depth were entered. Pre-programmed keyboard interrupt modes would not respond to subsequent inputs. Keyboard interaction was masked and additional dive plane commands could not be executed. The program could terminate only if the CTRL-BREAK key combination was executed.

From Reference 11 it was discovered that by removing the influence of the CTRL-BREAK character at the start of the programming code, programmed interrupts could be invoked.

By overcoming this impasse, program development was inevitable. Significant algorithm progression followed. "Canned" maneuvers, invoked by keyboard inputs, were developed and tested. Data collection and filing were implemented by actuating function key inputs.

The final phase of research was committed to the development and implementation of an Estimator/Observer control program. This algorithm would reconstruct the state variable, pitch angle, using depth and pitchrate sensor feedback. Bias created by the relatively inexpensive onboard sensors was determined by a Kalman filter and removed from the system. A cleaner, more robust control signal resulted.

B. A/D, D/A CONVERSION

The Analog to Digital (A/D) and Digital to Analog (D/A) conversion process is vital to the operation of this control system. Analog sensor signals from the AUV are digitized via the A/D process prior to entering the computer control program. Likewise, digitized command signals are converted to analog signals via the D/A process prior to return to the dive plane servos. For the preliminary AUV autopilot design described in Reference 7, the total cycle time for a complete I/O conversion process was 5 milliseconds with a 50 millisecond sample rate (20 Hz). In later paragraphs, discussion will focus on the need to decrease the sample rate in order to augment the larger, more complex control code.

The digitizing process is accomplished by the Data Translation (DT) board, DT 2801-A. It is installed internal to the computer as explained in Reference 7. The board configuration can be found in the technical reference manual [Ref. 14]. All conversions on the DT 2801-A board are controlled by writing data to, and reading data from, registers on the board. PCLAB subroutines is an interfacing software package that coordinates this task [Ref. 15]

Conversion of a voltage value to a digitized value depends on the number of bits of resolution used by the converter. For instance, a 12-bit converter has 4096 Number Of Codes (NOC). For a bipolar range of +/- 10 volts, a 0 NOC would correspond to -10 volts, while a 4096 NOC would correspond to +10 volts. An NOC of 2048, similarly, would correspond to 0.0 volts.

Before these digitized voltage values are usable in the software control program, they need to be scaled to an equivalent numerical voltage value. Equations found in

Appendix D of [Ref. 15] were implemented into the control code for such a reason.

C. REAL-TIME CONTROL CODE

The control program code was written in the programming language Turbo Pascal version 3.0. A more proficient version (V4.0) is presently available, but it is not currently supported by the interfacing software (PCLAB).

A real-time controller is "real-time" up to the limits of the software program implementing it. A significant advantage of Turbo Pascal is operation speed. Reducing large problems down into smaller, easier defined procedures affords faster execution. Each procedure can then be subdivided further, at the discretion of the programmer.

The control program executes an AUV dive control system within a user friendly multiple menu-shell setup. When running the compiled control program, the first screen encountered is the Main Menu. Here, the user is presented with a display of program titles and the options to quit or continue (run). The next screen offers a similar choice, where depressing the function key F1 starts the control mode. In this mode, the user is asked to name a "data file" (standard IBM DOS file). The results of the vehicle dynamics during the course of the run will be stored in this file. The final input screen prompts the user to "ENTER THE TARGET OPERATING DEPTH." Completion of this procedure activates the closed-loop dive plane control. The Run-Mode-Screen is introduced on the monitor displaying updated values of AUV depth, pitch, pitchrate, dive command angle, sensor bias, and attitude.

During closed-loop control, the user has four methods of operation:

- F3--exit from the "Active Control Mode," and file close-out.
- F2--activates a pre-programmed "Canned" maneuver.
- F1--exit from the "Active Control Mode," close-out previously selected file, creates new file with user interaction, and requests new target depth to reactivate closed-loop control.
- Any alpha/numeric key--temporarily exits from "Active Control Mode," prompts user for a new target depth, while retaining previously selected data file for data storage.

Two separate control programs were used to establish data collection and analysis. The first, AUVCOP3F.PAS, was designed as a closed-loop dive control system with full state feedback. The "COP3" corresponds to the third copy or generation control program. The "F" coincides with data Filing capabilities. "PAS" is Turbo Pascal's standard file type designation for programs to be complied. The second program, AUVCOP4F.PAS, implements an Observer/Estimator to reconstruct an unknown state variable from two sensor inputs. Sensor Bias is also estimated and removed from the system.

D. SAMPLE RATE CONTROL

A signal generator excites the DT 2801-A board to pass data to and from the data registers. The dial setting of the generator determines the rate at which this is accomplished. As mentioned earlier, a sample rate of 20 Hz was used in Reference 7.

When running the full-rate feedback control program (AUVCOP3F.PAS) at 20 Hz sample rate, Turbo Pascal computed the relatively small algorithm in 5 milliseconds. Manually reducing the sample rate during program runtime resulted in a stable controller as low as 3 Hz. An ideal condition, though, is to maintain a high sample rate to yield a more stable, faster responding controller.

The Estimator/Observer control program (AUVCOP4F.PAS), however, contains an additional 400 hundred lines of code. Here, computation time was increased significantly (greater than 150 milliseconds) and could not keep pace with the 20 Hz sample rate. To accommodate, the sample rate was reduced to 5 Hz. Likewise, new gains were derived.

Current hardware/software configurations of the AUV controller are limited in performing these multiple matrix calculations at high sample rates.

E. SOFTWARE LIMIT FOR ACTUATOR SATURATION

Present hardware configurations on the 30 inch model required small input voltages to actuate the full range of the dive plane angle. During previous research of open-loop tests, dive plane angle was compared with the corresponding input voltage. Results concluded that a linear proportionality on the order of 0.01 volts per degree dive plane angle existed. This was true for a range of 0.0 to 10.0 degrees. Hence, the operating range of the dive planes became +/- 10 degrees, and respectively, the operating voltage range became +/- 0.1 (using the hand-held transmitter). The dive plane voltage range of +/- 0.1 was implemented into the analog computer configuration. The control code was changed by creating a software stop prior to sending the final command signal to the plane. This was implemented by limiting the final signal sent through the D/A conversion to +/- 0.1 volts in the form,

```
if abs(u) > 0.1  then  u = 0.1*abs(u)/u.
```

F. FILES FOR DATA RECOVERY

During the open-loop vehicle tests, data acquisition was achieved by using the software program DT Notebook [Ref. 5] on the IBM PC/AT. The computer's system configuration file had to be specifically structured for DT Notebook operation. However, the programming language Turbo Pascal used an entirely different configuration. The two programs, therefore, could not be run simultaneously. In other words, data acquisition by DT Notebook could not be performed while running the closed-loop control program.

An alternate method of data recovery was devised using Turbo Pascal [Ref. 11]. The keyboard commands for data file use are found in Chapter IV.C. Although additional throughput is needed for data file simulation, the time used in writing data to a hard disk was found to be negligible in comparison to the sample rate.

V. CLOSED-LOOP RESULTS

A. GENERAL

In Figure 5.1, the closed-loop full state feedback system is displayed in block diagram convention. It is comprised of the system model:

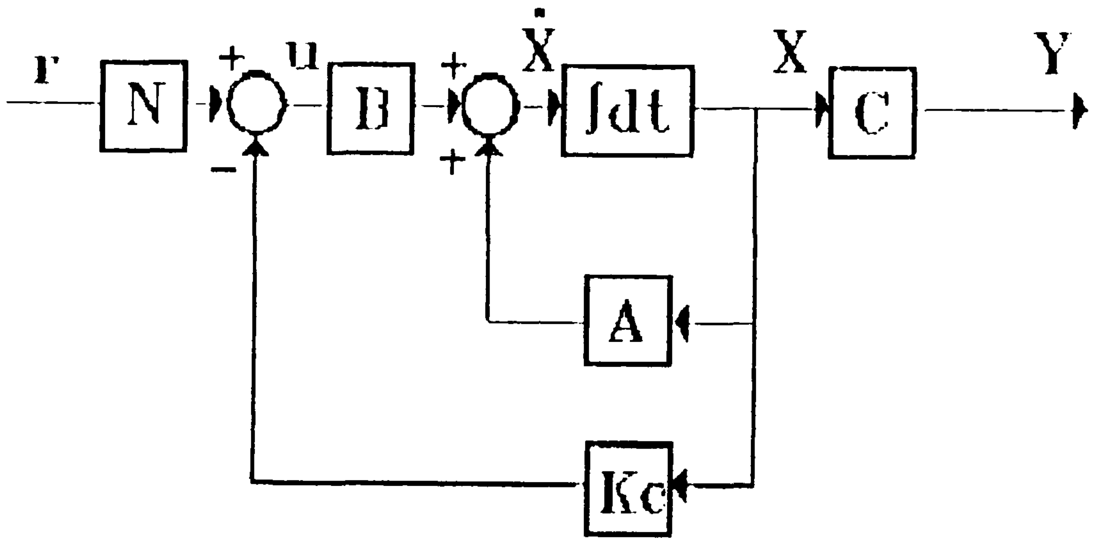


Figure 5.1 Full State Feedback Diagram

$$\dot{\underline{x}} = [\underline{A}]\underline{x} + [\underline{B}]\underline{u}$$

and the feedback control:

$$\underline{u} = [\underline{N}]\underline{r} - [\underline{Kc}]\underline{x}$$

where the input of target depth is the set point $\underline{r}$, the closed loop gains are $\underline{Kc}$, and the setpoint matrix $\underline{N}$ is designed for zero steady state depth error. The following sections in this chapter evaluate the response of the above system under varying conditions. The gains are fixed and the sample rate is 20 Hz.

B. RESULTS OF FULL STATE FEEDBACK

In the full state feedback controller, the closed loop gains are designed for an AUV speed of 2.1 Ft/s. In the following figures, AUV depth voltages ranging from 0.0 to 10.0 volts represent the water depth from the surface to a depth of 10.0 feet below the surface respectively. The dive command voltage ranges from +/- 0.1 volts equivalent to a dive plane angle of +/- 10 degrees. In Figure 5.2, a target depth of 6.0 volts becomes the new set point from 1.0 volt. Subsequently, a dive command of 10 degrees is sent to the dive planes. Almost immediately, the AUV responds and levels out at the new depth. The entire "run" lasts approximately 120 sample increments or six seconds, as expected, with an overall dominant time constant close to the 1.0 second used in the gain design process.

C. EFFECT OF ACTUATOR SATURATION

As previously mentioned, the dive plane actuator is limited to a range of +/- 10 degrees. If this limit is removed then the results of Figure 5.3 are found. Comparing

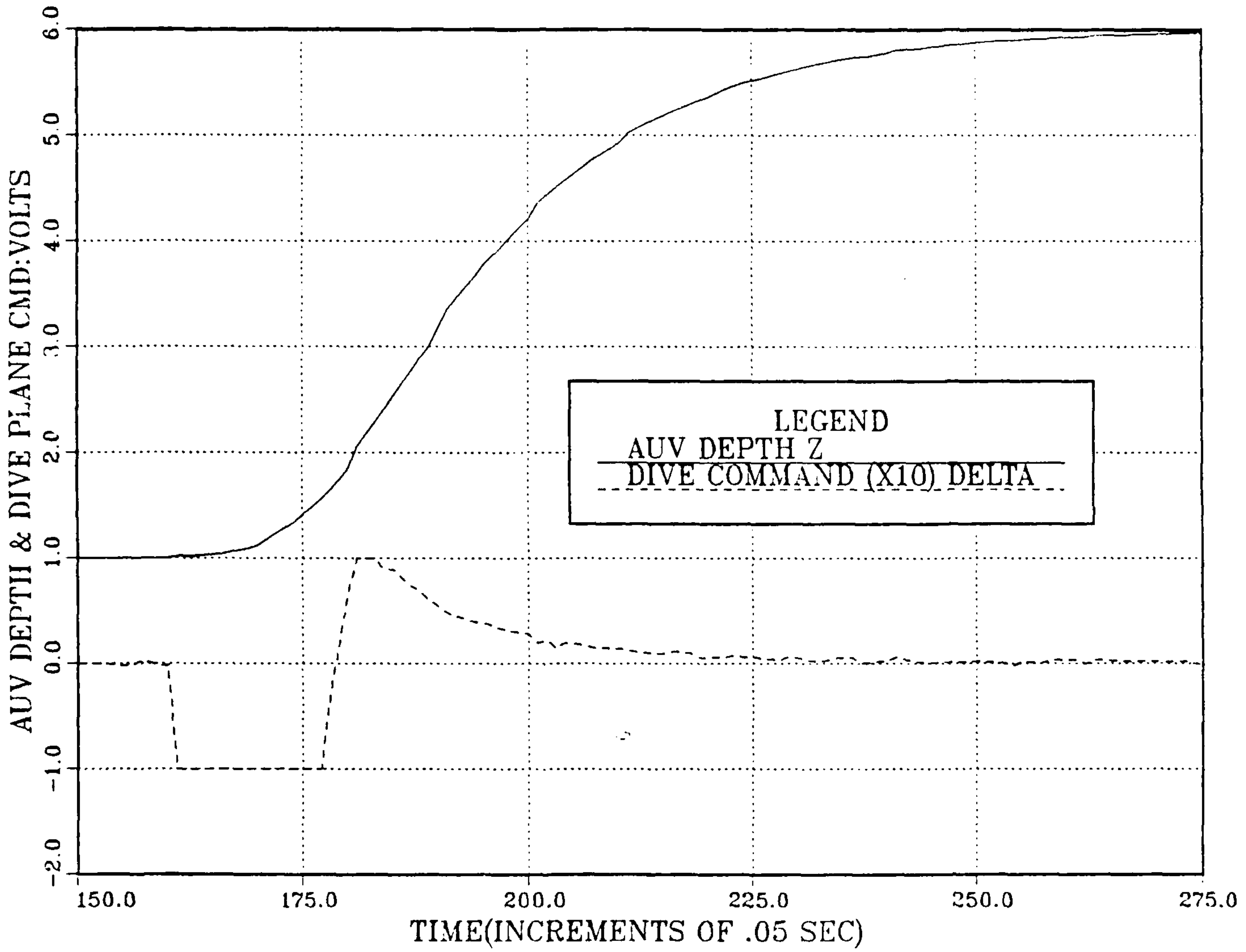


Figure 5.2 Depth Maneuver from 1 to 6 Feet at a Speed of 2 Ft/s and 20 Hz Sample Rate

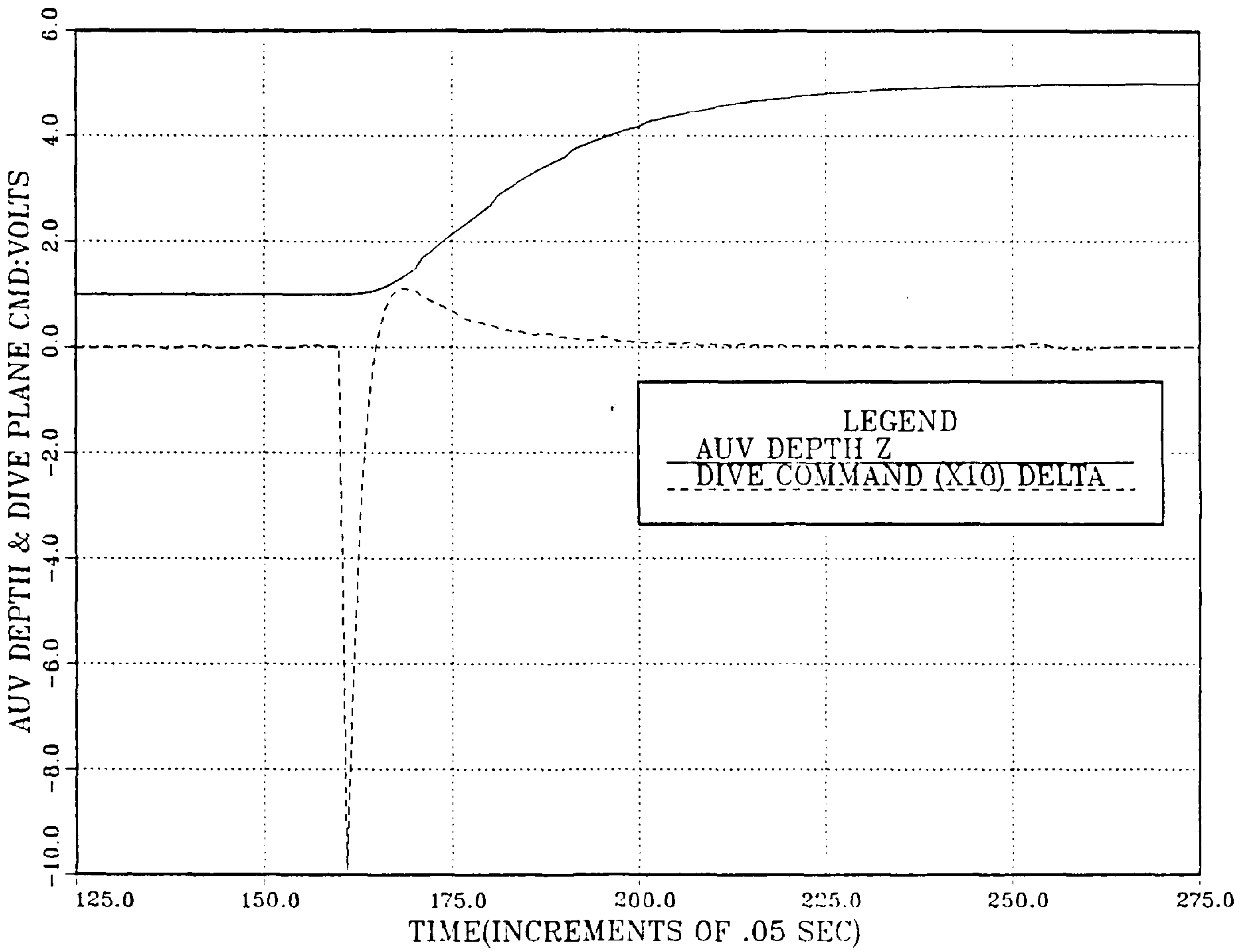


Figure 5.3 Depth Maneuver from 1 to 5 Feet without Saturation on the Dive Planes. Speed: 2 Ft/s at 20 Hz Sample Rate

this to a second run, having actuator saturation, the results of Figure 5.4 show that the non-saturated dive command achieves a slightly faster AUV depth response. Figure 5.4 attains 5.0 volts in approximately six seconds whereas Figure 5.3 attains target depth in approximately 5.5 seconds. In Figures 5.5 and 5.6, a similar comparison is made. Here, however, the depth change is considerably smaller. Similarly, the response time difference is less noticeable.

As one would expect, a larger actuator in the dive plane drives a quicker AUV response. While actuator rate-of-change limits may be an additional limit to consider, these results assumed that such a limit was not present.

D. EFFECT OF VEHICLE SPEED

Though the closed loop gains were designed to operate the AUV at a speed of 2 FT/s, other speeds were tested to determine the robustness of the controller so obtained. In Figures 5.7, 5.8, and 5.9, the vehicle speeds of two, four, and six FT/s respectively were evaluated when the AUV depth changed from six to four volts. In Figure 5.7, the AUV smoothly reaches the assigned depth without overshoot in approximately five seconds. In Figure 5.8, target depth is attained in nearly 3.5 seconds with a slight overshoot before leveling out. Finally, in Figure 5.9, overshoot is most noticeable, but target depth is accomplished in close to three seconds. Based on these observations, it can be

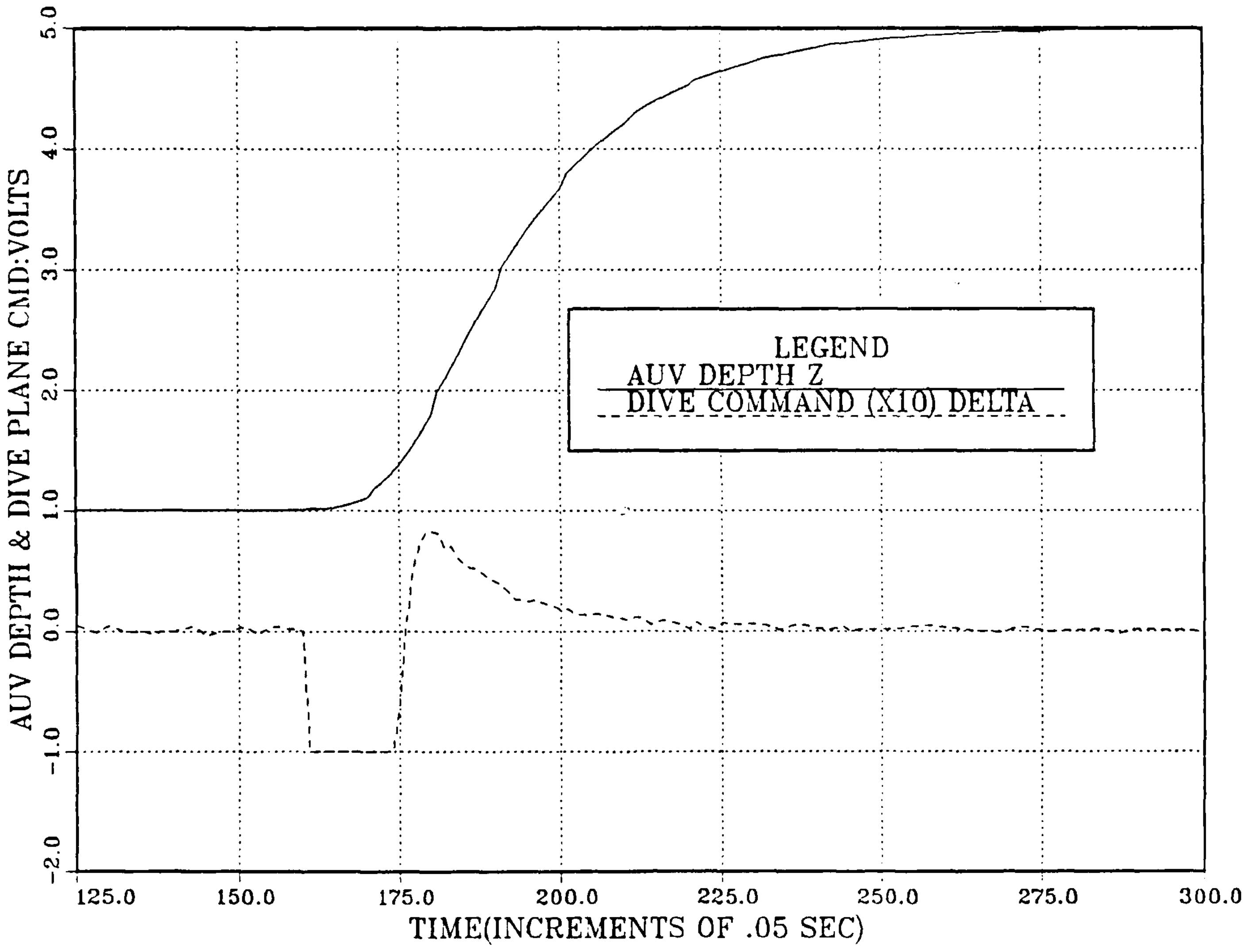


Figure 5.4 Depth Maneuver from 1 to 5 Feet with Saturation at 10° Dive Plane Angle. Speed: 2 Ft/s at 20 Sample Rate

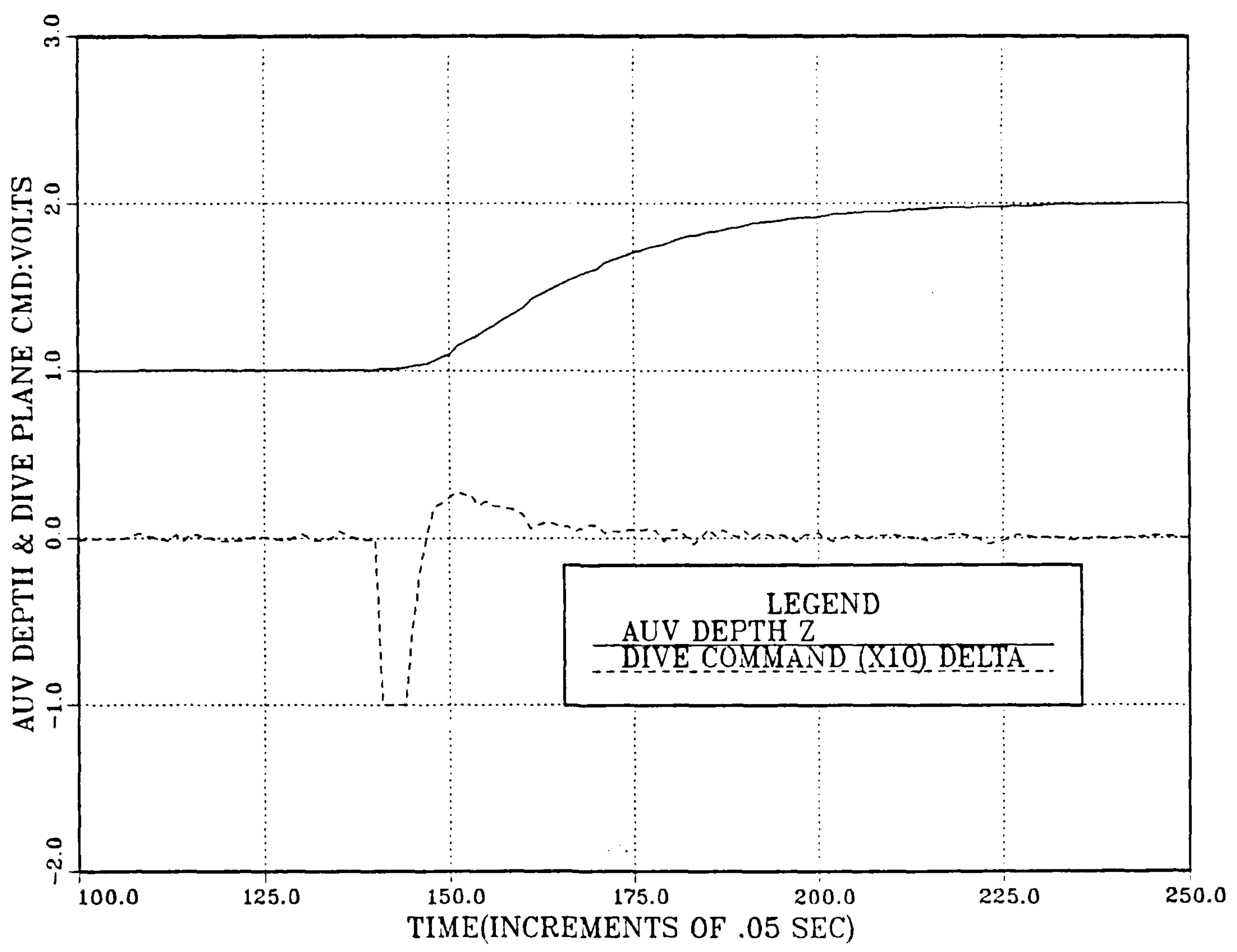


Figure 5.5 Depth Maneuver from 1 to 2 Feet with Dive Plane Saturation. Speed: 2 Ft/s at 20 Hz Sample Rate

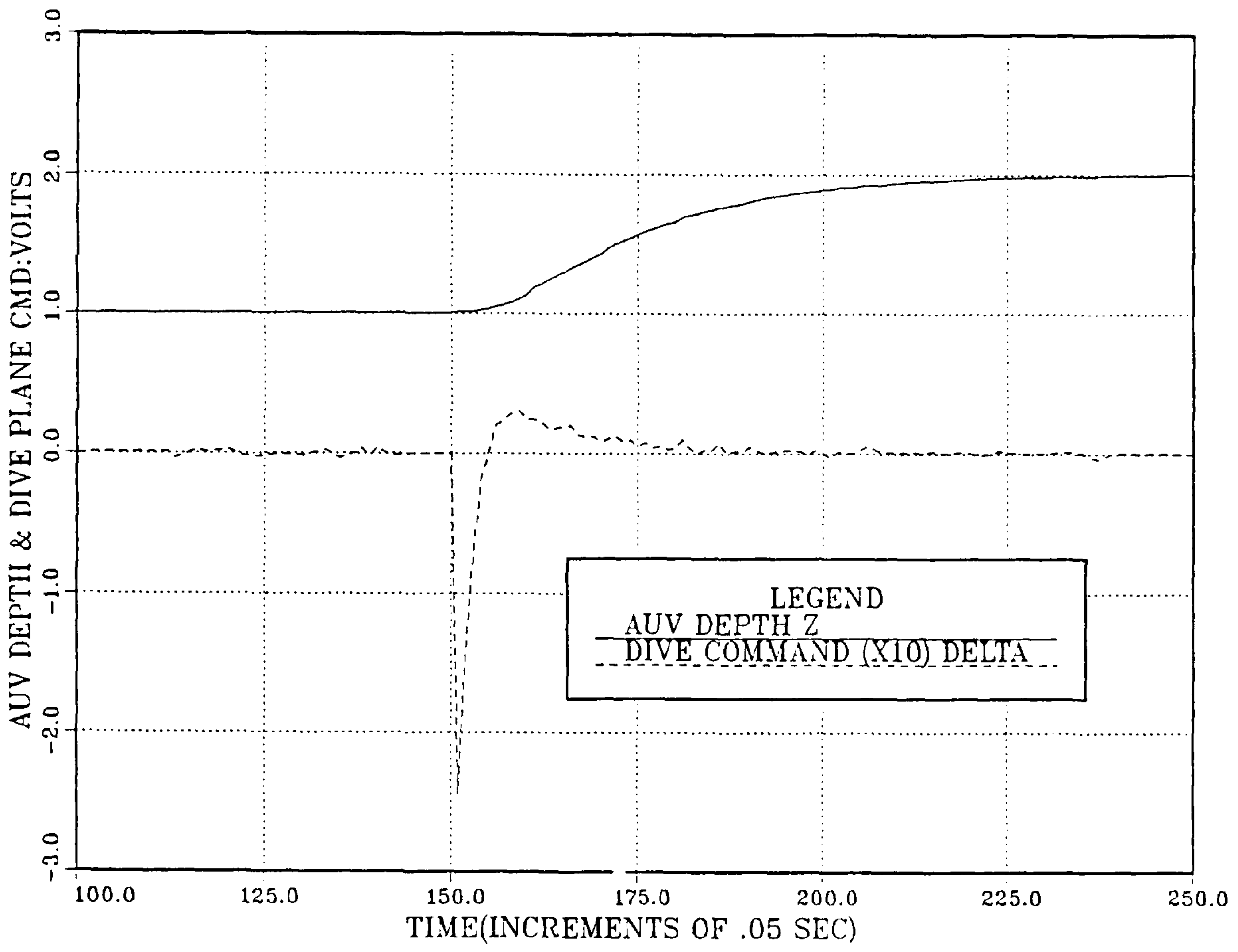


Figure 5.6 Depth Maneuver from 1 to 2 Feet without Dive Plane Saturation. Speed: 2 Ft/s at 20 Hz Sample Rate

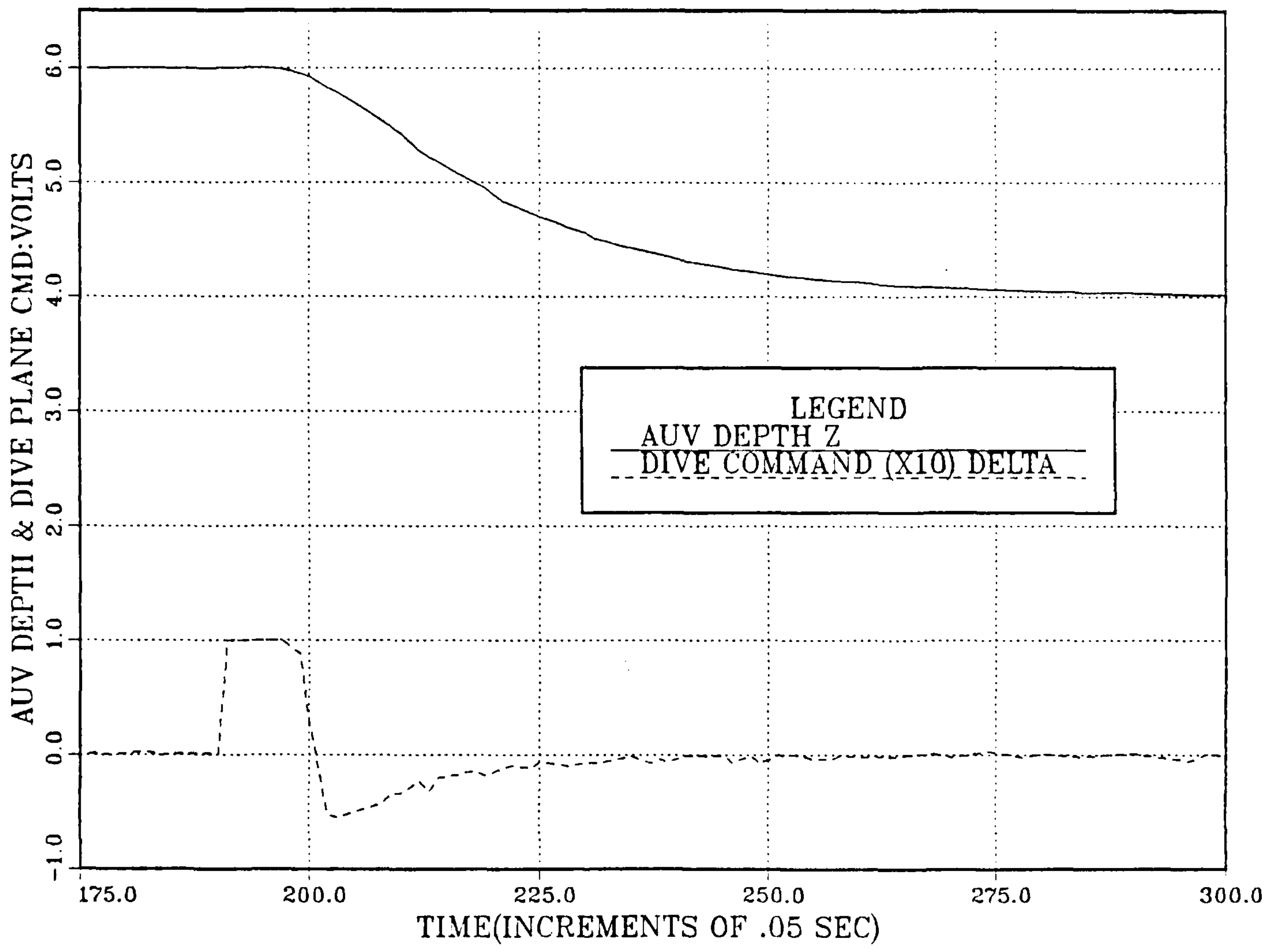


Figure 5.7 Depth Maneuver from 6 to 4 Feet at a Speed of 2 Ft/s and 20 Hz Sample Rate

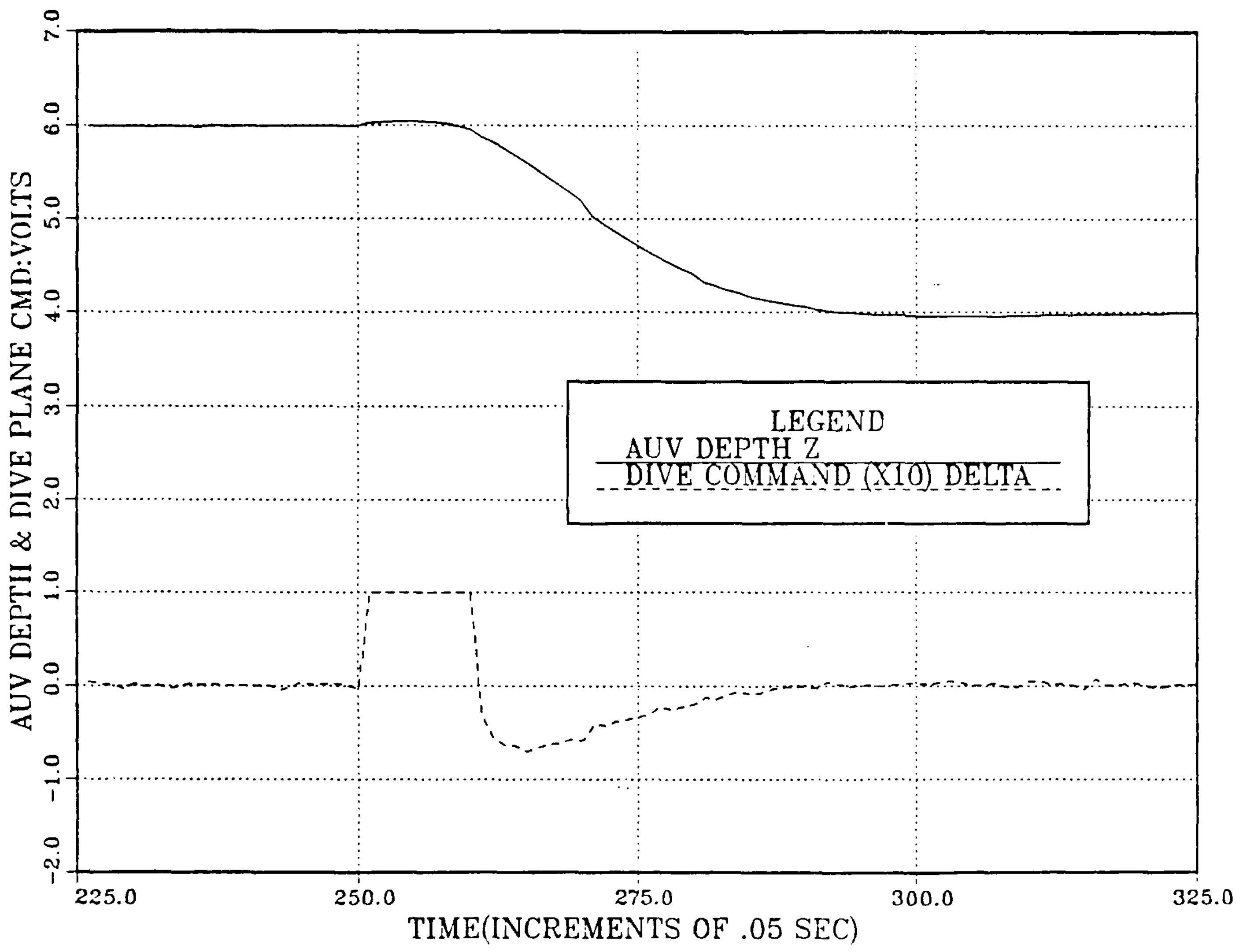


Figure 5.8 Depth Maneuver from 6 to 4 Feet at a Speed of 4 Ft/s and 20 Hz Sample Rate

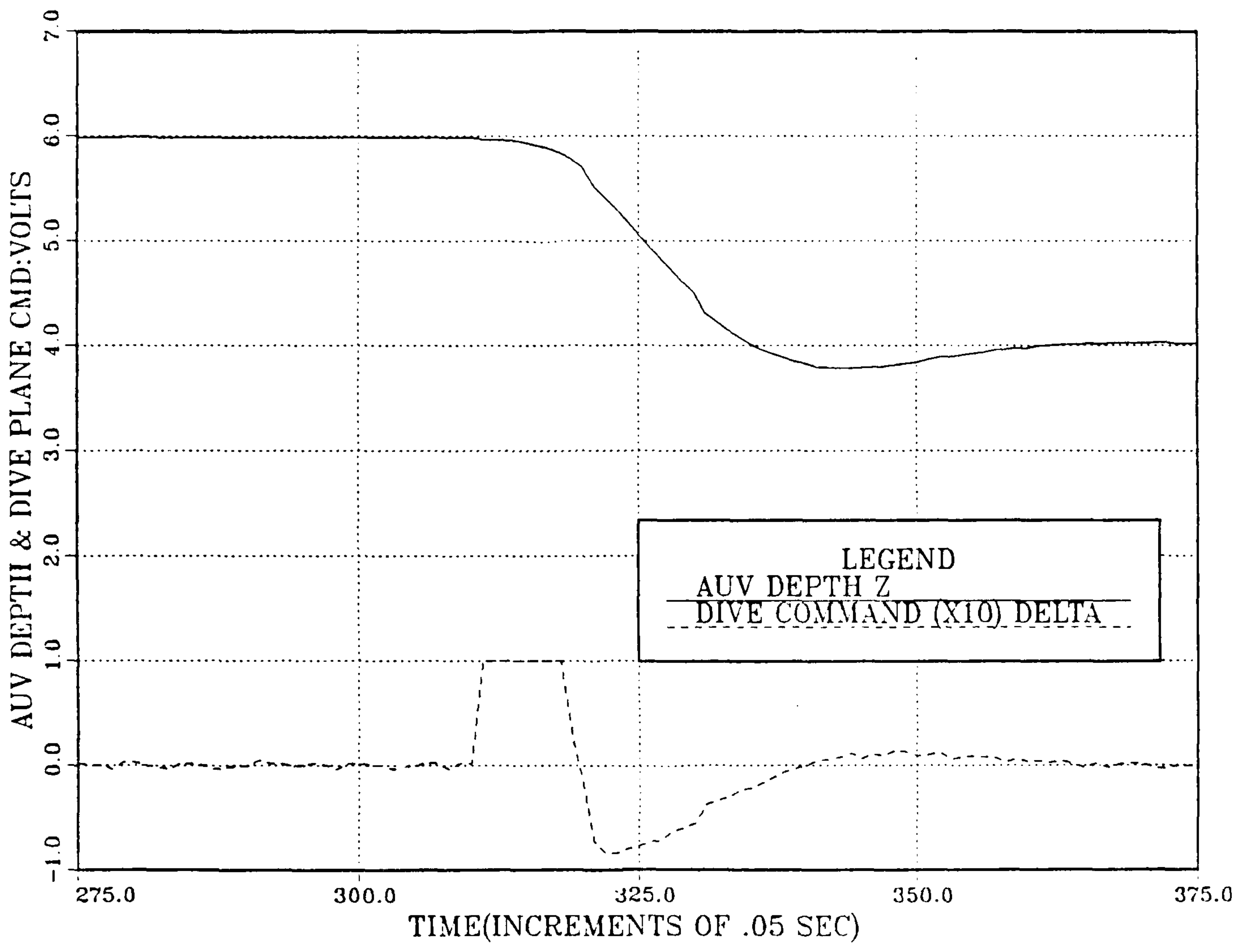


Figure 5.9 Depth Maneuver from 6 to 4 Feet at a Speed of 6 Ft/s and 20 Hz Sample Rate

noted that through a 300% change in the speed parameter, the closed loop control performance remains stable with only a 40% change in response time and, for overshoot, a change of less than 15%.

VI. BIAS ESTIMATION AND RESULTS

A. GENERAL

In this chapter, the final stages of program design are implemented. The results are graphed and evaluated. Here, the fullstate feedback controller in Chapter V is combined with a parameter estimator and state observer system as shown in Figure 6.1. The appendix details this program code.

The parameter estimator determines sensor bias of the depth pressure cell and pitchrate gyro. The three-state observer determines the missing state, pitch angle, and filters the return signal of the remaining two states.

With the inclusion of the above algorithms (over 400 lines of additional code), the 20 Hz sample rate or 0.05 seconds per sample, had to be reduced. Significant increase in computation time prevented a sample rate much higher than 5 Hz or 0.2 seconds per sample. Hence, the new sample rate became 5 Hz and the poles in the z-plane were changed accordingly. The remaining figures in this chapter show the results of the new code at work. For purposes of evaluation, vehicle depth change remains constant while other parameters are varied.

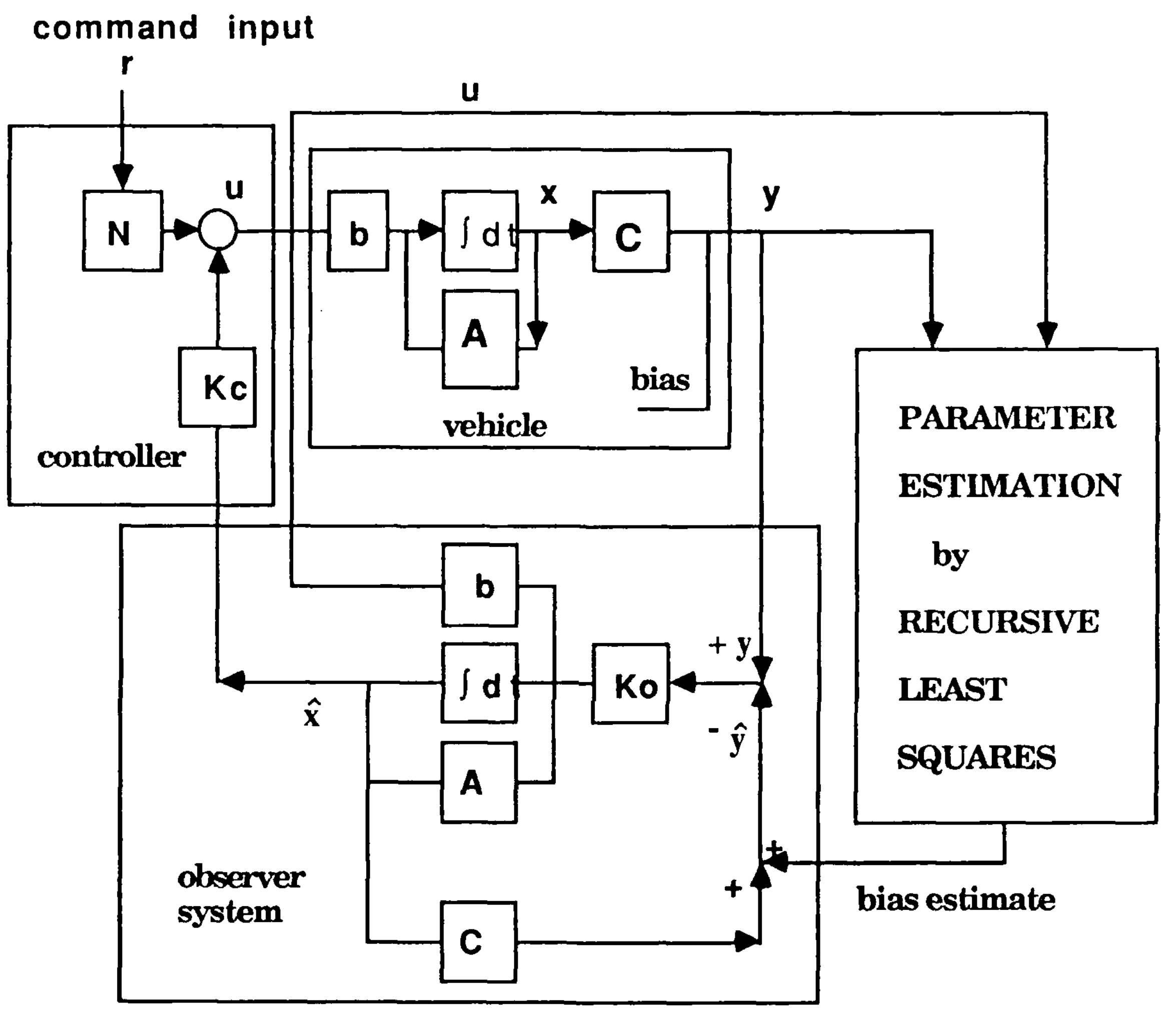


Diagram of the Bias Compensated Model Based Controller

Figure 6.1 Diagram of the Bias Compensated Model Based Controller

B. EFFECT OF SPEED CHANGE ON AUV RESPONSE

In Figure 6.2 at an AUV speed of 2 ft/sec, a smooth dive maneuver from 1.0 volt to 5.0 volts yields a response time of 13 seconds, as compared to six seconds in Figure 5.4. The decrease in sample rate has moved the poles, retarding AUV response time by more than a factor of two.

In Figure 6.3, the vehicle speed is doubled to four ft/sec and the same maneuver is invoked. Here the depth response curve is not as smooth or direct, but no overshoot is observed. Since the poles are designed for a slower speed, the vehicle tries to settle out prematurely, delaying the depth change process. This maneuver was accomplished in 17 seconds.

Results of this vehicle speed increased to six ft/sec are shown in Figure 6.4. Again, the curve is not smooth, with abrupt settling occurring after two-thirds of the depth change and then continuing on to 5.0 volts depth. The completion time for this maneuver is greater than 20 seconds.

For each increase in speed, maneuver completion time is extended, but target depth overshoot, however, does not occur.

C. AUV RESPONSE TO A BIAS SIGNAL

As mentioned in preceding chapters, the return signals from the relatively inexpensive sensors onboard the 30 inch model are prone to include sensor bias. In the subsequent

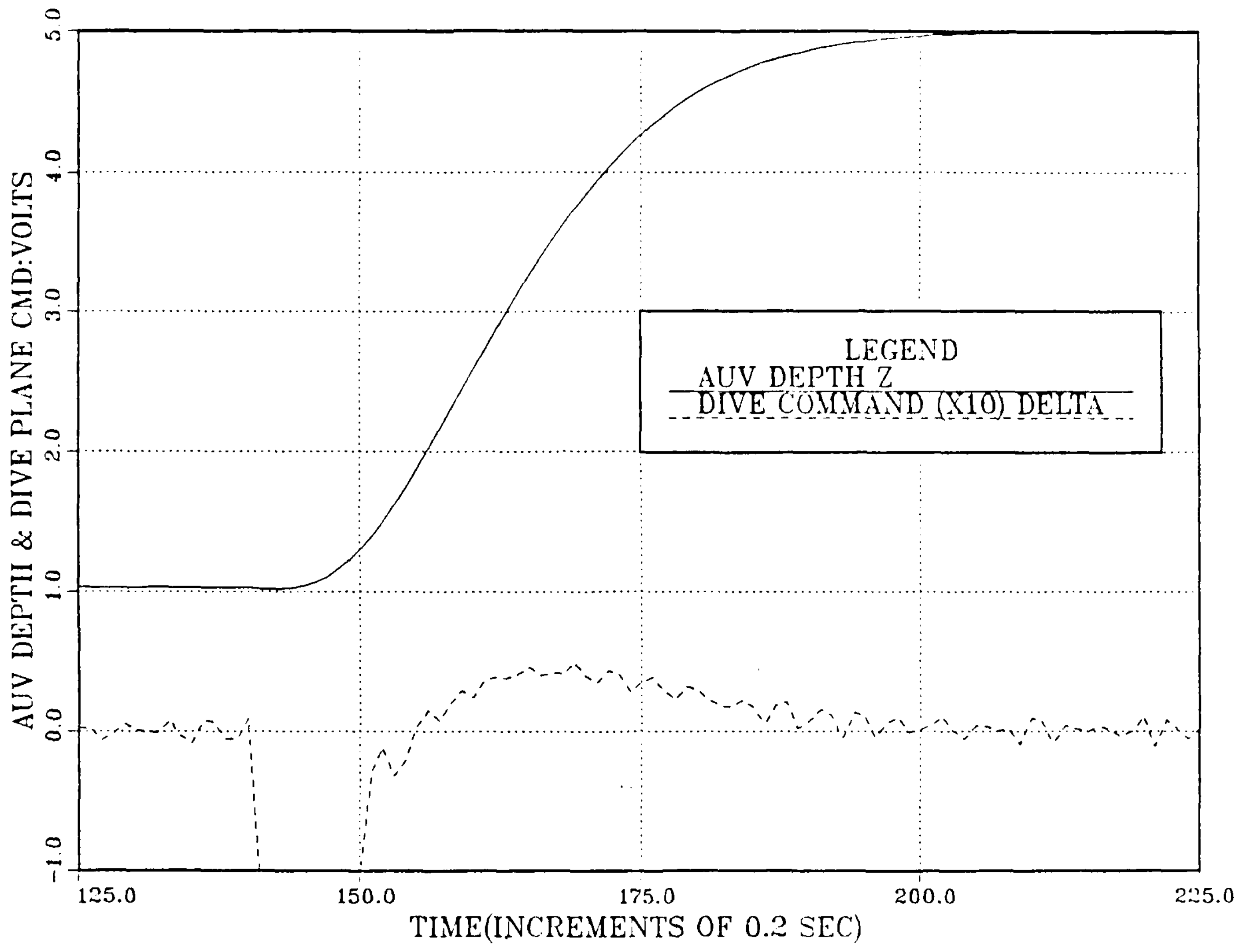


Figure 6.2 Depth Maneuver from 1 to 5 Feet at a Speed of 2 Ft/s and 5 Hz Sample Rate

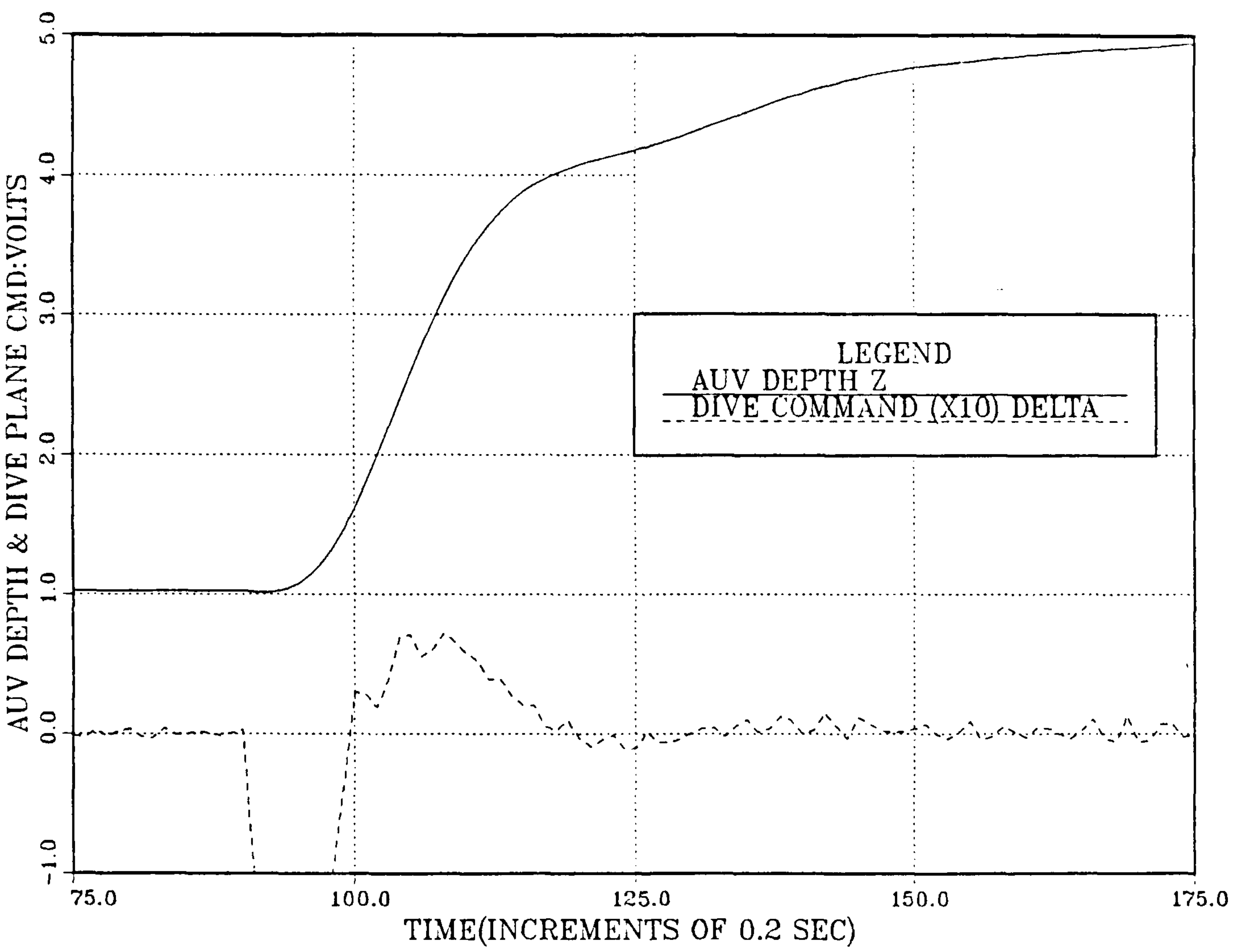


Figure 6.3 Depth Maneuver from 1 to 5 Feet at a Speed of 4 Ft/s and 5 Hz Sample Rate

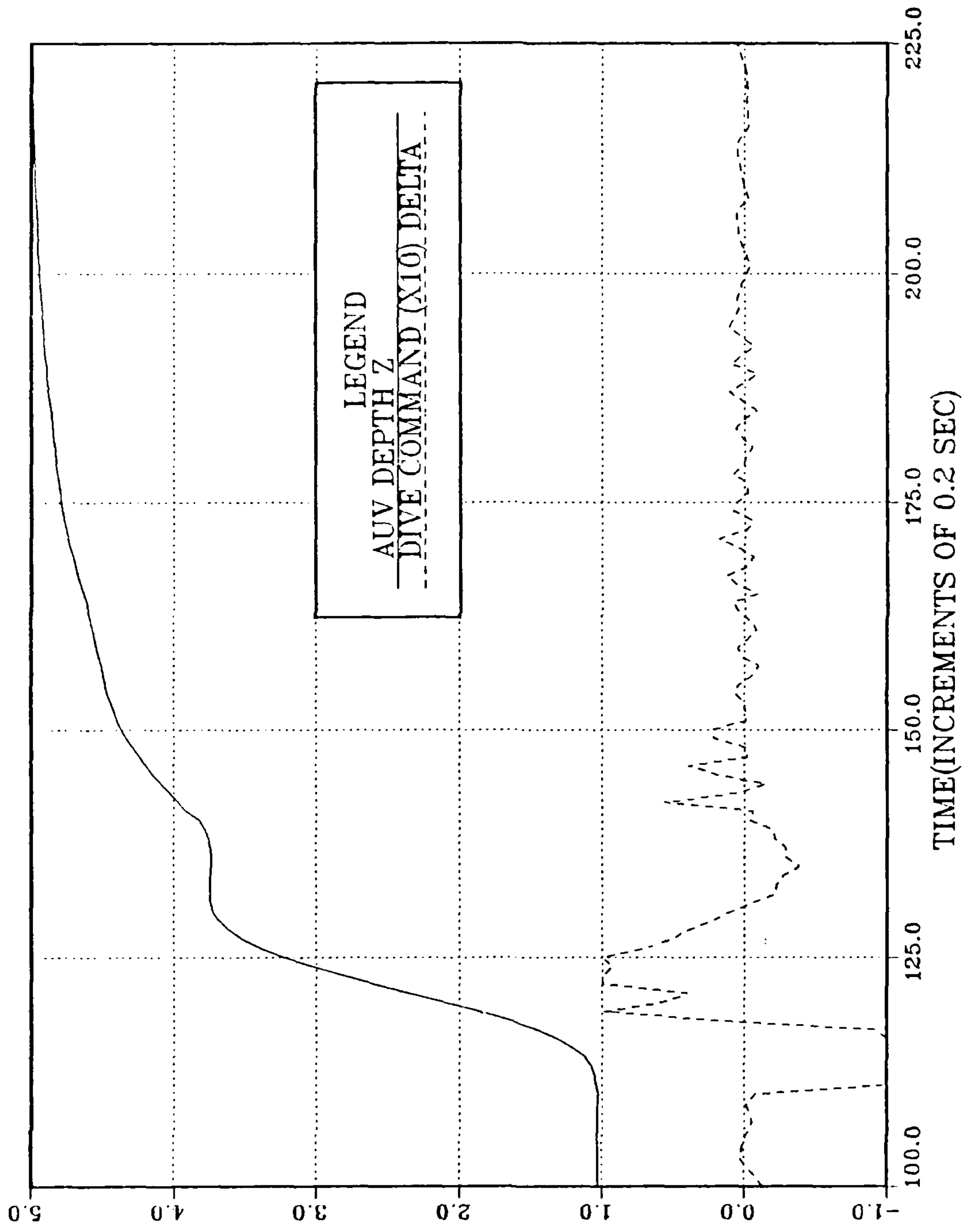


Figure 6.4 Depth Maneuver from 1 to 5 Feet at a Speed of 6 Ft/s and 5 Hz Sample Rate

figures, the results of vehicle depth with sensor bias is shown. A 0.1 volt sensor bias in Figure 6.5 results in a corresponding 0.1 volt decrease in actual depth as compared to the target depth of five volts. Although maneuver completion time is not influenced (equivalent to Figure 6.2 with zero bias), the target depth is never quite reached. Similarly, in Figure 6.6, a larger sensor bias of 0.2 volts results in a vehicle depth falling short of the target depth by the same amount.

By slightly adjusting the program code, the estimated bias can be removed from the system resulting in a cleansed depth response signal, shown in Figures 6.7 and 6.8.

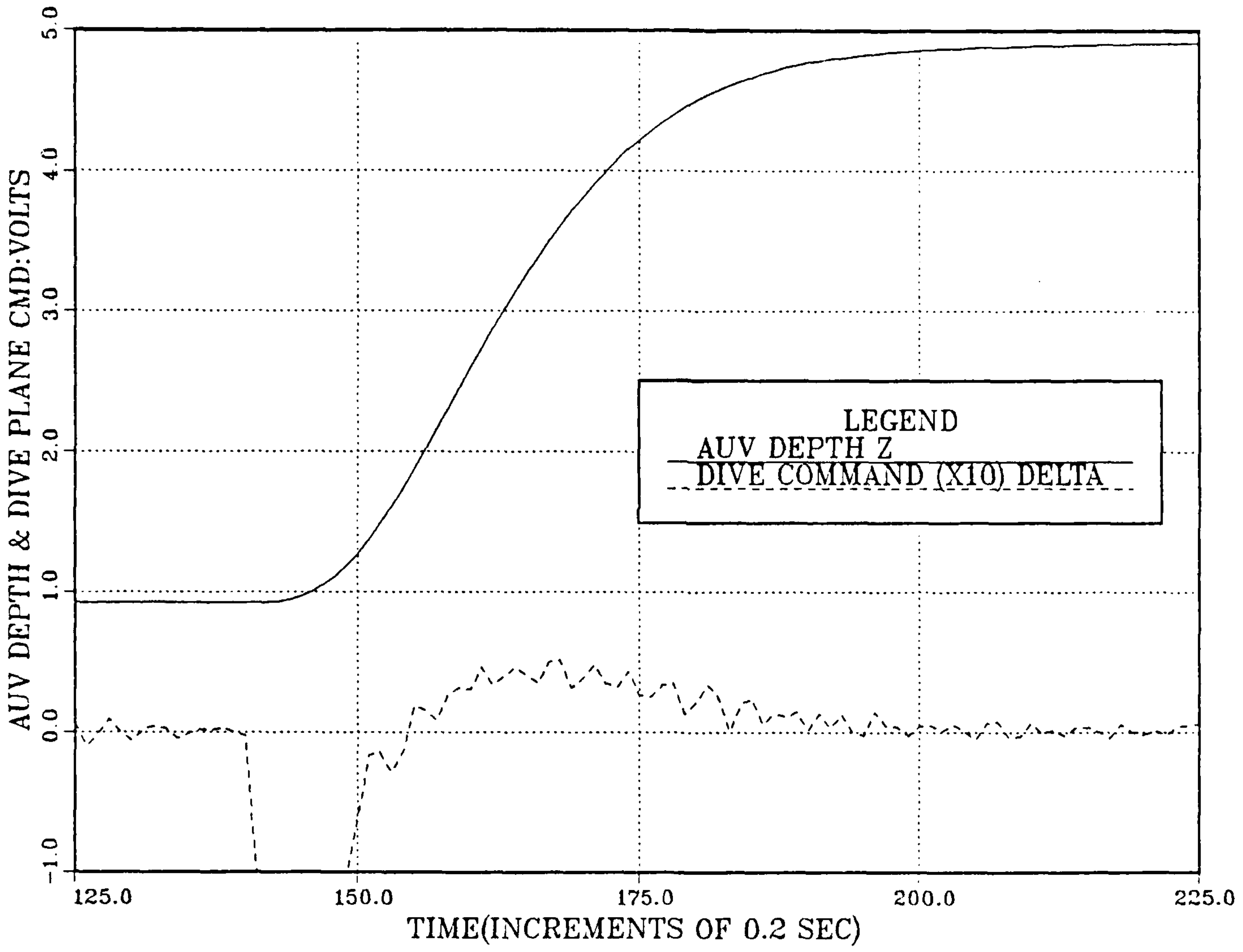


Figure 6.5 Depth Maneuver from 1 to 5 Feet with .1 Volts Sensor Bias, Speed 2 Ft/s and 5 Hz Samrle Rate

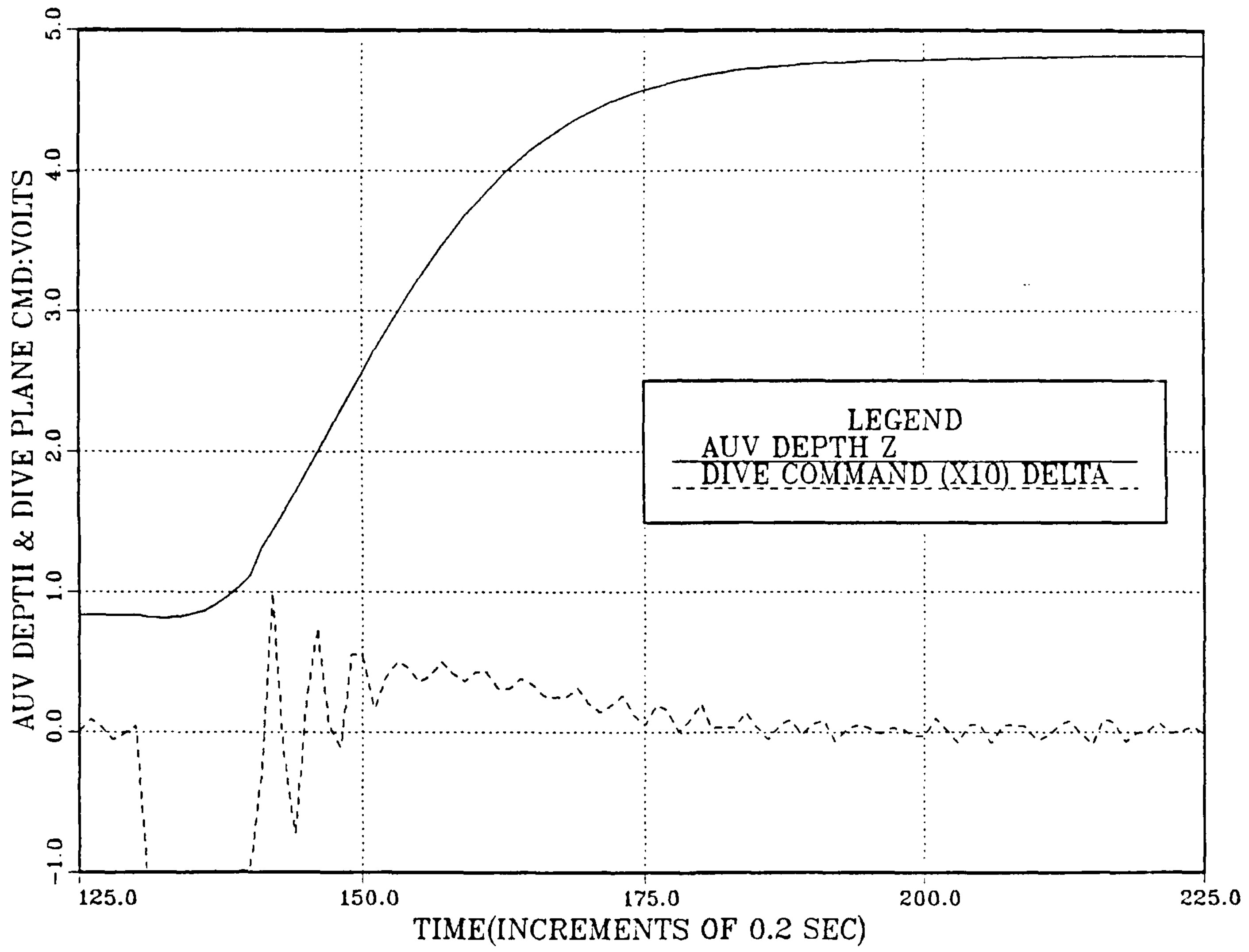


Figure 6.6 Depth Maneuver from 1 to 5 Feet with .2 Volts Sensor Bias. Speed 2 Ft/s and 5 Hz Sample Rate

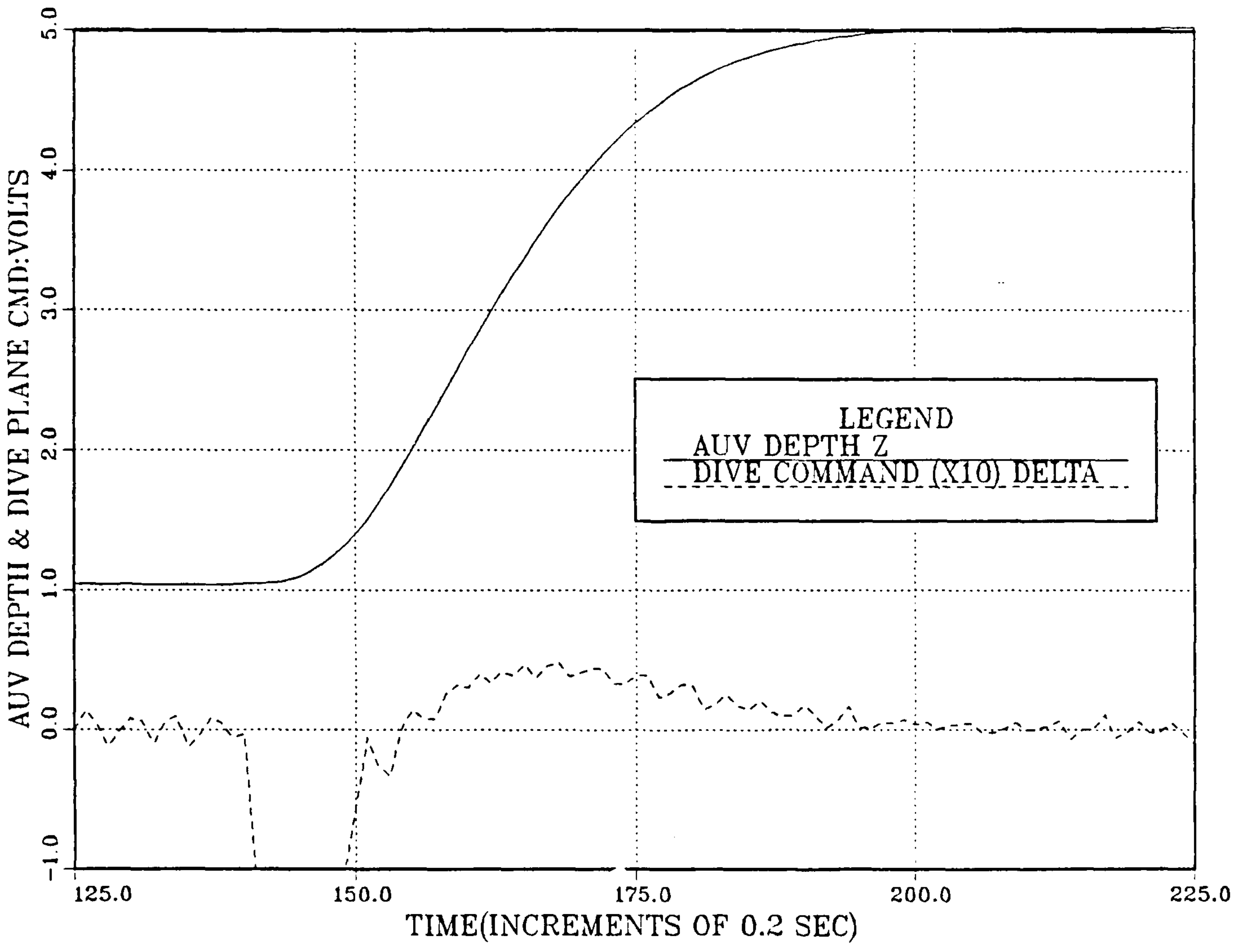


Figure 6.7 Depth Maneuver from 1 to 5 Feet, Corrected for a Sensor Bias of .2 Volts. Speed: 2 Ft/s and 5 Hz Sample Rate

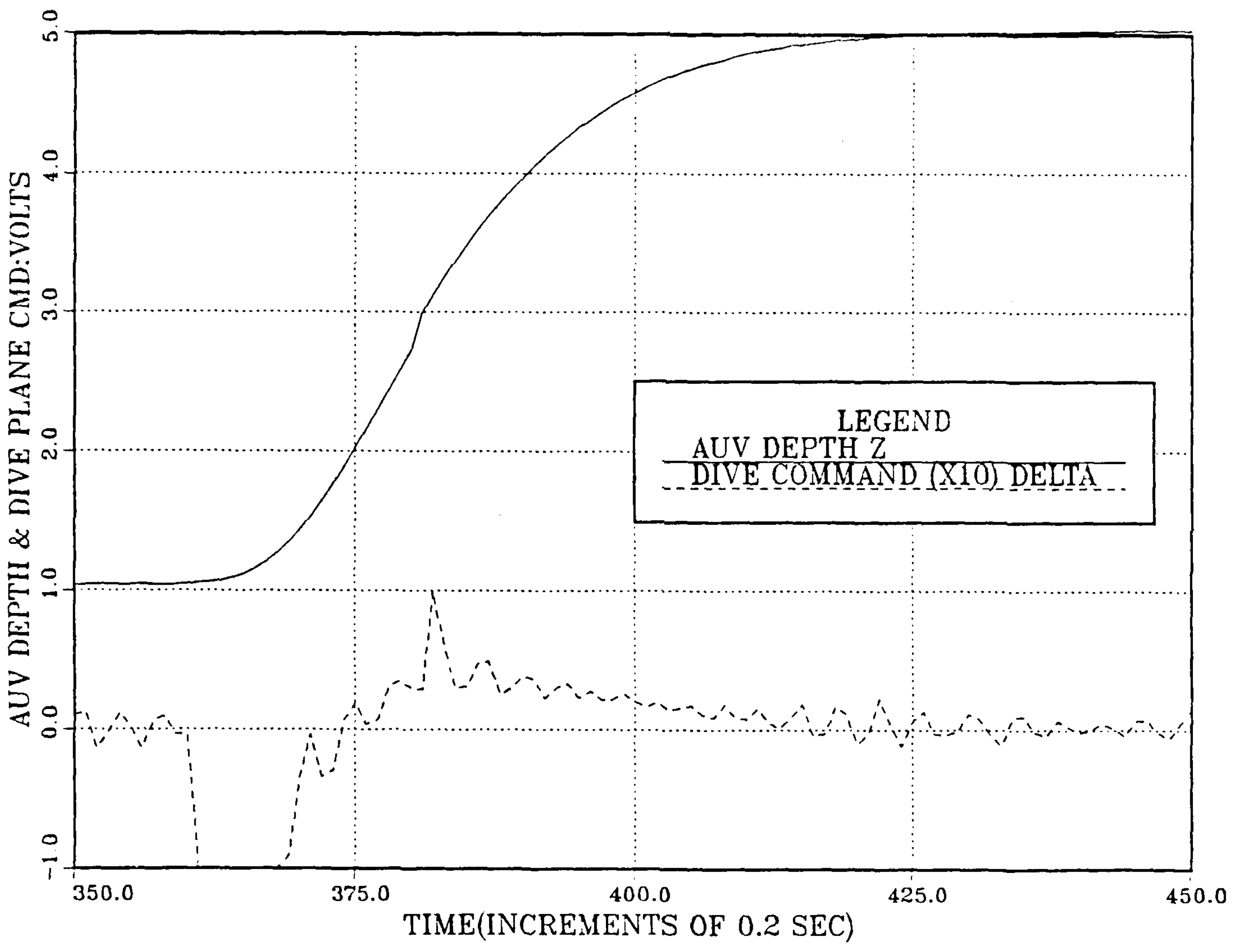


Figure 6.8 Depth Maneuver from 1 to 5 Feet, Corrected for a Sensor Bias of .1 Volts. Speed 2 Ft/s and 5 Hz Sample Rate

VII. SUMMARY AND CONCLUSION

A. SUMMARY

This thesis presents the implementation of a real time autopilot dive plane controller for the NPS AUV testbed. The controller includes a state observer and sensor bias estimator. The approach to the design of the autopilot includes:

- Manipulation of vehicle dive dynamics to develop linearized equations of motion,
- Implementation of an Analog Simulator using data from open loop control. This affords more effective and efficient development of the full state and partial state feedback control algorithms,
- Hardware/Software interface via A/D and D/A conversion boards,
- Program development using Turbo Pascal version 3.0,
- Sample rate control on program code of differing magnitudes,
- Incorporating data recovery via file opening techniques.

B. CONCLUSION

This study developed a dive plane autopilot for the NPS testbed that showed signs of robustness. Under closed loop control, the vehicle stabilized at various depths under wide ranges of speed with no overshoot.

As program code increased in size, throughput increased reducing sample rate, and therefore, reducing autopilot response time. Future research of the steering control autopilot may need to optimize current algorithms to prevent further throughput increase and to optimize robustness characteristics. Parallel-tasking processors may be an option to large, complex, programming code, creating a faster throughput.

APPENDIX

CONTROL CODE FOR AUTOPILOT

```
{$c-}
program   AuvAutoPilot  ( input, output );

{ TITLE       : Autonomous Underwater Vehicle Auto Pilot Program.
  AUTHOR      : LT Jerry J. Reina USN
  APPLICATION : real-time controller with a Model Based Compensator to
                estimate pitch angle, and a parameter estimator to determine
                sensor bias
  DATE        : 8 Dec 1988

  Project Description : This program implements digital control of the NPS
  autonomous underwater vehicle (AUV) in the vertical or dive plane.It samples
  vehicle sensor input from two channels : depth and pitchrate.  The Model
  Base Compesator determines pitch angle.  All states are then multiplied by
  the controller gains and compared to the target depth.  An appropriate dive
  command is determined and sent back to the vehicle via D/A conversion boards.
  A deflection on the dive planes result. The sample rate is 5 Hz.}

{ GLOBAL DECLARATIONS     -------------------------------------------------- }

const
{ ----------- Screen declarations --------------- }

   x1          = 5;                     { Upper left corner : left edge }
   y1          = 2;                     { Upper left corner : upper edge }
   x2          = 75;                    { Lower right corner : right edge }
   y2          = 24;                    { Lower right corner : bottom edge }

type
  str10 = string [10];
  str60 = string [60];

var
  hr,hr2,min,min2,
  sec,sec2,hun,hun2            : byte;
  seconds                      : real;
  option, controlmode,
  reply,reply2                 : char;

{------------------------- INCLUDED FILES Declarations   ------------------ }


{$I pcldefs.tp  }   { PC LAB Trubo Pascal routines.                          }
{$I pclerrs.pas }   { PC LAB error code messages file.                       }

{$Idrawbox2.auv}
{ Input x1,y1,x2,y2 : integer to specify the corner limits of the box.
This procedure clears screen and draws a rectangular box of specified
dimension using ASCII double line characters.                                }

{$Iclrbox2.auv}
```

```
{ Input x1,y1,x2,y2 : integer to specify the corner limits of the box.
  This procedure uses a FAST means of clearing a box of specified dimension.
  The box dimension should be delcared as constants.                      }

{$Iboxprint.auv }
{ Input the printrow, leftboxedge, rightboxedge  : inceger and printstring :
  str60.  This procedure centerprints the string in the box at the printrow
  specified without overwriting the box border.                           }

{$Ishowfast.auv}
{ Input message : str60, column,row : integer.  To specify the x,y position
on the screen for a FAST message print.                                   }

{$Ikeyhit.auv}
{ This is a boolean function which returns true or false if key is pressed;
it also returns keycode replies VAR reply, reply2 : char   .              }

{$Itabxy.auv}
{ Input tabcol,tabrow : integer; like gotoxy                              }

{$Iboxpause.auv}
{ Input xpause,ypause : integer to specify where "Press any key to continue"
  message is to be printed.                                               }

{$Igetkey.auv }
{ Input as a string of chars, the set of acceptable replies; ie 'YyNn'.  This
  procedure waits until one of the acceptable replies has been entered.   }


{ *****************  MAIN PROGRAMS PROCEDURES   *************************** }
{ ***************** USER INTERFACE MODULES      *************************** }
procedure MainMenu ( var reply : char );

{  This procedure presents the AUV screen and solicits an option to Run
   the AUV from the StatusAndCommand procedure or to Quit.               }

begin
   repeat
     clrscr;
     drawbox2(x1,y1,x2,y2);
     boxprint(y1+3,x1,x2,'N A V A L   P O S T G R A D U A T E   S C H O O L');
     boxprint(y1+5,x1,x2,'D E P A R T M E N T   O F ');
     boxprint(y1+6,x1,x2,'M E C H A N I C A L   E N G I N E E R I N G');
     boxprint(y1+8,x1,x2,'AUTONOMOUS   UNDERWATER   VEHICLE');
     boxprint(y1+10,x1,x2,'DIGITAL   AUTOPILOT  CONTROL   PROGRAM');
     boxprint(y1+12,x1,x2,'****************************************');
     boxprint(y1+15,x1,x2,'Do You want to RUN this program ..');
     boxprint(y1+16,x1,x2,'or Do You want to QUIT and return to DOS ?');
     boxprint(y1+20,x1,x2,'>>>>  ENTER Q OR R  <<<<');
     getkey ('QqRr',reply,reply2);
   until ( reply in ['Q','q','R','r'] ) and (reply2 = chr(0));
end;
```

```
procedure StatusAndCommand ( var mode : char );
{ This procedure begins the control program  screen.                        }

var
   mode2                 : char;

procedure StatusAndCommandScreen;
{  This is the status and control screen and solicits a user input of F1 to
   RUN the program or Q to Quit and exit to the main menu.                   }

begin        { --------------- StatusAndCommandScreen ------------------- }
     clrbox2 (x1,y1,x2,y2);
      boxprint(y1+1,x1,x2,'AUV STATUS / COMMAND AND CONTROL SCREEN');
      boxprint(y1+2,x1,x2,'==========================================');
      boxprint(y1+7,x1,x2,'CHOOSE YOUR DESIRED CONTROL MODE  :');
      boxprint(y1+9,x1,x2,'ENTER KEY  << F1 >>   TO START AUV CONTROL');
      boxprint(y1+11,x1,x2,'ENTER  << Q >> TO QUIT AND RETURN TO MAIN MENU');
      boxprint(y1+16,x1,x2,'PRESS EITHER   F1   OR  Q');

end;         { ---------------- StatusAndCommandScreen ------------------ }

{ ********************** CLOSED LOOP CONTROL ROUTINES ******************* }

procedure ClosedLoopControl;
{ This module comprises the closed loop control scheme.                      }
label 5;
LABEL 1;
const
     maxdepth        = 33;
     mindepth        = 0;
     updateincrement = 10;

type
    try = array[1..10] of real;
   activecontrolmode   = ( run, reset, exit );
   allowabledepthrange = mindepth..maxdepth ;

   auvattitude         = ( climb, maintain, diving );

   digitalintegerarray  = array [1..3] of integer;

var
filename:string[14];
filevar:text;
   auvdepth, auvdepthvolts,auvspeed,auvpitch,auvpitchvolts,auvpitchvoltsxx,
   auvspeedvolts, auvpitchrate, auvpitchratevolts,estdepth,err,
   deptherrorvolts, targetdepthvolts ,bias1,divevolts,tgtnew,targetdepth,speed:r
eal;
   adv                                              : digitalintegerarray;
   j,status,time                                    : integer;
   modereply,modereply2                             : char;
   activemode                                       : activecontrolmode;
   updatecounter,initial                            : integer;
   depthrange                                       : allowabledepthrange;
   attitude                                                   : auvattitude;
   ac1                                                        :real;
   ac2                                                        :real;
   ac3                                                        :real;


procedure GetTargetDepth (var tgtdepth : real ;
                          var tgtdepthvolts : real );

{ This procedure solicits the target AUV operating depth and converts it to
  an AUV equivalent targetdepth analog voltage and passes both of these
  parameters. }


begin          { --------  GetTargetDepth ---------------------------- }
    clrbox2 (x1,y1,x2,y2);
    boxprint(y1+10,x1,x2,'ENTER THE A U V TARGET OPERATING DEPTH');
    boxprint(y1+11,x1,x2,'NOTE : THE DEPTH SHOULD BE IN VOLTS (0.0 - 10.0)');
    repeat
       begin
          boxprint (y1+13,x1,x2,'ENTER THE TARGET OPERATING DEPTH ');
          gotoxy (x1+33,y1+15);
          read ( tgtdepth );
       end;
    until  tgtdepth >= 0.0 ;

    tgtdepthvolts :=   tgtdepth ;

end;           { --------  GetTargetDepth ---------------------------- }

procedure RunModeScreen;

{ This procedure displays the Closed Loop Control Screen in the RUN MODE.   }

begin         { -------------- RunModeScreen ------------------ }
      clrbox2 (x1,y1,x2,y2);
      boxprint(y1+1,x1,x2,'A U V   S T A T U S   /  C O N T R O L  S C R E E N')
;
      boxprint(y1+2,x1,x2,'==============================================');
      boxprint(y1+4,x1,x2,'STATUS OF A U V OPERATING PARAMETERS :');

      write (tabxy (x1+5,y1+6),'AUV DEPTH     (volts)  : ');

      write (tabxy (x1+5,y1+7),'AUV PITCH     (volts)  : ');


      write (tabxy (x1+5,y1+8),'AUV PITCHRATE (volts)  : ');

      write (tabxy (x1+5,y1+9),'AUVDIVEPLANE  (volts)  : ');
      write (tabxy (x1+5,y1+10),'BIAS =       :');
      boxprint(y1+11,x1,x2,'A U V  CONTROL STATUS :');

      write (tabxy (x1+5,y1+13),'CURRENT TARGET DEPTH : ');
      write (tabxy (x1+5,y1+14),'CURRENT MODE         : ');
      write (tabxy (x1+5,y1+15),'CURRENT MANEUVER     : ');

      boxprint(y1+18,x1,x2,
          'PRESS KEY   F1 .. TO ENTER NEW TARGET DEPTH.        ');
      boxprint(y1+19,x1,x2,
          'PRESS KEY   F2 .. TO STOP ACTIVE CONTROL AND RESET.');

      boxprint(y1+20,x1,x2,
          'PRESS KEY   F3 .. TO EXIT ACTIVE CONTROL.           ');
      end;          { --------------- RunModeScreen------------------ }

procedure UpdateRunModeScreen (updatedepth,updatepitch,updatepitchrate
,updatedivevolt : real;
                                   updatetargetdepth : real;
                                   updatemode: activecontrolmode ;
                                   updateattitude : auvattitude;
                                   var biasl: real);
{  This module updates the Closed Loop Control Run Mode Screen with updated
   display parameters. Updates occur in intervals specified by updateincrement
   interval declared in ClosedLoopControl procedure.                       }

begin       {  --------  UpdateRunModeScreen ----------------------------  }

{    UPDATES   STATUS OF A U V OPERATING PARAMETERS                }
   writeln (tabxy (x1+37,y1+6),updatedepth:6:3);
   writeln (tabxy (x1+37,y1+7),updatepitch:6:3);
   writeln (tabxy (x1+37,y1+8),updatepitchrate:6:3);
   writeln (tabxy (x1+37,y1+9),updatedivevolt:6:3);
   writeln (tabxy (x1+37,y1+10),biasl:8:6);
{ UPDATES THE   A U V  CONTROL STATUS                               }
   write (tabxy (x1+30,y1+13),updatetargetdepth:6:2);
   case updatemode of
      run  : writeln (tabxy (x1+30,y1+14),'RUN  ');
      reset: writeln (tabxy (x1+30,y1+14),'RESET');
      exit : writeln (tabxy (x1+30,y1+14),'EXIT ');
   end;
   case updateattitude of
      maintain : writeln (tabxy (x1+30,y1+15),'MAINTAINING DEPTH       ');
      climb    : writeln (tabxy (x1+30,y1+15),'CLIMBING TO TARGET DEPTH');
      diving   : writeln (tabxy (x1+30,y1+15),'DIVING TO TARGET DEPTH  ');
   end;

end;        {-----------UpdateRunModeScreen-----------------------------------}

procedure GetDigitalSensoryData ( var depthanalogvolts,pitchanalogvolts,
                                        pitchrateanalogvolts :real) ;

{  This procedure uses PCLAB routines to sample selected input telemetry
   channels from the AUV and digitizes these inputs and multiplies them
   by the specified gains.

   DT 2801-A / DT 707 Board set up:   channel 1  - AUV depth input
                                      channel 2  - AUV pitch input
                                      channel 3  - AUV pitchrate input    }
```

```
const

{ These are AUV to DT 2801-A / DT 707 hook up board channel configurations,
      conversion and computational arguments.                                 }

     depthchannel   = 1;        { AUV output to DT-707 input channel assignment  }
     depthpfs       = +10.0;    { Peak depth signal value                        }
     depthmfs       = -10.0;    { Minimum depth signal value                     }

     pitchchannel   = 2;        { AUV output to DT-707 input channel assignment  }
     spdpfs         = +10.0;     { Peak pitch signal value                        }
     spdmfs         = -10.0;     { Minimum speed signal value                     }

     pitchratechannel = 3;      { AUV output to DT-707 input channel assignment  }
     pitchratepfs   = +10.0;    { Peak pitchrate signal value                    }
     pitchratemfs   = -10.0;    { Minimum pitchrate signal value                 }

     noc            = 4096;     { Number of Codes; conversion resolution.
                                  The DT 2801-A performs a 12 bit conversion.
                                  NOC = (2 ^ conversion bits), ie 4096           }

{ SetUpAdc and ADConTrigger PCL function arguments
                                                : p 6-8 PCL  documentation  }

     boardnum       =1;

     numa2dchan     = 3;
     timingsource   = 2;        { -- Sets a external trigger,internal clock }
     adcgain1       = 1;
     adcgain2       = 2;
     adcgain4       = 4;                  { Sets the A/D gain; 1,2,4,8 are options
}
     adcgain8       = 8;
     startchannel   = 1;
     endchannel     = 3;

var
   pitchadv,
   depthadv,
   pitchrateadv,
   signaladv,                                        { Signal analog data value      }
   counter,status,
   chanum,i,j                  : integer;

begin              { ------------ procedure GetDigitalTelemetry ---------- }

{ Set up the DT 2801-A board to take data.                                   }

   status := SelectBoard (boardnum);

{ Set up the DT 2801-A board to take data from 3 input channels; Data sampling
  is initiated by the ADConTrigger single channel sample of the depth channel
  and then single ADC value samples of the speed and pitchrate follow.
  The Trigger is connected to the DT 707 board at terminal 49 from a signal
  generating source.                                                    }

           status := ADConTrigger ( depthchannel, adcgain1, depthadv );
           status := ADCValue ( pitchchannel, adcgain1, pitchadv );
           status := ADCValue ( pitchratechannel, adcgain1, pitchrateadv );


{ Convert the digitized Analog Data Values for pitch, depth, pitchrate to
  analog voltage values.  The algorithm for this conversion is found in
  Appendix D of the PCLAB documentation.                              }

   depthanalogvolts := ( depthadv * (depthpfs-depthmfs)/noc )
                                                        + depthmfs;
   pitchanalogvolts := ( pitchadv * (spdpfs-spdmfs)/noc )
                                                        + spdmfs;

   pitchrateanalogvolts := ( pitchrateadv *
                                   (pitchratepfs - pitchratemfs)/noc )
                                                        + pitchratemfs;

{   status := Terminate;     }     { --- PCL routine, closing the DT 2801-A   }

end;              { ------------ procedure GetDigitalTelemetry  ---------- }

procedure ATTITUDE_ ( tdepthvolts, adepthvolts : real;
                       var attitude : auvattitude   );


const

   depthcontroltolerence =  0.1;
   modelgain             =  1.0;      { This simulates a model referenced }
                                      {  gain parameter                   }
var

    voltsdifference      : real;


begin       { ------ Errorvolts ------------------------------------------ }

   voltsdifference := tdepthvolts - adepthvolts;


{ +++++++++++++++++ Control voltage filter   ++++++++++++++++++++++++++++ }

{ These conditions check if depth is within tolerence.  }

   if  ( voltsdifference > 0) and
           ( abs(voltsdifference) <= depthcontroltolerence ) then
      begin
         attitude := maintain;
```

```
        end
     else if  ( voltsdifference < 0) and
              ( abs(voltsdifference) <= depthcontroltolerence ) then
        begin
           attitude := maintain;
        end

{ This condition checks if actual depth is less than target + tolerence.}

     else if  ( voltsdifference > 0) and
              ( abs(voltsdifference) > depthcontroltolerence ) then
        begin
           attitude := diving;
        end

{ This last condition checks to see if the actual depth is more than target +
  tolerence.   }

      else if  ( voltsdifference < 0) and
              ( abs(voltsdifference) > depthcontroltolerence ) then

        begin
           attitude := climb;
        end;


end;          { ------ Errorvolts ------------------------------------------ }


procedure EST(var up,yg,z:real;var vhat,pitch:real;var initial:integer;
                                 var bias,a1,a2,a3: real);
  {This procedure uses a Model Based Compensator to determine pitch angle.
  A parameter estimation using RLS fit determines sensor bias.}

        label  3;
 type
        try = array[1..10] of real;
        trytry = array[1..10,1..10] of real;
        try40 = array[1..40,1..40] of real;
        tryt40 = array[1..40] of real;

  const
           ns = 1;

  var
         beta,g,khat,xdata,g2,phid,phig,fc,phi2,khat2,xhat2,xhat,
         g3,h3,khat3,phi,phat,xu,xy,xz,xs,xnew: try;
         spt,f,spt2,f3,spt3 : trytry;
         nx,np,ny,n1,nf,n2,nsnf,time,i,j : integer;
         zd,zf,zf2,uf,yf,fs,af,y,yhat,ud,u,yd,p0,sr3,sr2,sr,sq3,sq,sq2,
         zdd,sg2,sigmax,gbias: real;
```

```
procedure initializeArrays;
     begin
          for i := 1 to 10 do
          begin
               beta[i]:=0.0;
               khat[i]:=0.0;
               xhat[i] := 0.0;
               phi[i] := 0.0;
               xu[i]:=0.0;
               xy[i]:=0.0;
               xz[i]:=0.0;
               xnew[i]:=0.0;
          end;
          np := 2*ns;
          nx := np + 1;
          ny := 1;
          y:= 0.0;
          uf:= 0.0;
          ud:=0.0;
          yd:=0.0;
          yf := 0.0;
          zd := 0.0;
          zf := 0.0;
          time := 0;
          {sigmax = variance of the bias}
          sigmax := 0.1;
          sq := sigmax;
          sq2:= sigmax;
          sq3 := sigmax;
          sr := 1.0;
          sr2 := 1.0;
          sr3 := 1.0;
          p0 := 1.0e6;

          for i:= 1 to nx do
          begin
               for j:= 1 to nx do
               begin
                    spt[i,j] := 0.0;
                    spt2[i,j] := 0.0;
                    f[i,j] := 0.0;
                    f3[i,j]:=0.0;
                          if i = j then
                          begin
                          spt[i,j]:= p0;
                          spt2[i,j]:=p0;
                          spt3[i,j]:=p0;
                          f[i,j]:= 1.0;
                          end;
               end;
               g[i]:= 0.0;
               g2[i]:= 0.0;

          end;
```

```
                  g[nx]:= 1.0;
                  g2[2]:= 1.0;
                { fc[1]:= 2.9272E-2;
                  fc[2]:= -6.9013E-1;
                  fc[3]:= 1.3633E-1;
                  fc[4]:= -7.5566E-2;
                  fc[5]:= 0.0;
                  fc[6]:= 9.8861E-2;
                  fc[7]:= 3.9545E-1;
                  fc[8]:= 5.9317E-1;
                  fc[9]:= 3.9545E-1;
                  fc[10]:= 9.8861E-2;}
                  fc[1]:=0.80000;
                  fc[2]:=0.20000;
                  nf:= 1;
                  fs := 5.0;
                  f3[1,1]:=0.6977;      {discretized A-matrix}
                  f3[2,1]:=0.1680;
                  f3[2,2]:=1.0;
                  f3[3,1]:=-0.0178;
                  f3[3,2]:=-0.20;
                  f3[3,3]:=1.0;
                  g3[1]:=0.4251;        {discretized B-matrix}
                  g3[2]:=0.0451;
                  g3[3]:=-0.0031;
{                  khat3[1]:=-0.0073;khat3[2]:=-0.1046;khat3[3]:=0.4250;}
                  {gains for 5Hz.,poles@[0.75 0.76 0.77] kob7 }
                  khat3[1]:=0.0;khat3[2]:=-0.4029;khat3[3]:=0.5677;
                  {kob8 @ ob8poles=[.7 .71 .72] at 1ft/s}
                  bias:=0.0;
             end;

procedure innerprod(var y9: real;var phi9,x9:try;var nx9:integer);


    var
          j: integer;
           begin
          y9:= 0.0;
             for j := 1 to nx9 DO
             begin
                  y9 := phi9[j] * x9[j]+ y9;
             end;
           end;

procedure update(var phi8: try;var u8,y8: real; ns:integer);



   var
         n2:integer;
         i: integer;
         phisave: try;
```

```
          begin
                 n2:=ns*2+1;
                 for i:=1 to n2 do
               begin
                 phisave[i]:=phi8[i];
               end;

                 i := ns;
                 while i >= 2 do
                       begin
                          phi8[i]:= phisave[i - 1];
                          phi8[ns + i] := phisave[ns+i-1];
                          i := i - 1;
                       end;
                       phi8[1]:= y8;
                       phi8[ns + 1] := u8;
                 end;

       procedure filter(var u4,y4:real;var x4,fc4:try;var nf4:integer);
                 var
                    nf42 : integer;
                    savex4:try;

                    begin
                         update(x4,u4,y4,nf4);
                         nf42:= 2 * nf4;
                         innerprod(y4,x4,fc4,nf42);
                    {writeln('inside filter after inner,y4,x4=',y4,x4[1],x4[2],x4[
3]);
}
                    end;

       procedure newest(var nx7: integer;var  xhat7,khat7,h7: try;
                             var   y7:real);
              var
                 delta: real;
                 i,j: integer;
                 xsave:try;

                 begin
                  for i:=1 to nx7 do
                     begin
                     xsave[i]:=xhat7[i];
                     end;
                 delta := y7;
                      for j:= 1 to nx7 do
                       begin
                          delta:= delta - h7[j] * xhat7[j];
                       end;
                       for i:= 1 to nx7 do
                       begin
                          xhat7[i] := xsave[i] + khat7[i] * delta;
                       end;
                 end;
```

```
    procedure hholder(var nr6,nc6: integer;var m16: try40);
        var
          l,col,i,j,k: integer;
           temp,sigma,sum : real;
           d6: tryt40; {array[1..40] of real;}
           c6,h6:try40;{ array[1..40,1..40] of real; }

  begin
                  for col:= 1 to nc6 do
        begin
                    sum := 0.0;
                    for j := col to nr6 do
                    begin
                        sum:= sum + m16[j,col]*m16[j,col];

                    end;
                    sigma:= sqrt(sum);
                    for j:= col to nr6 do
                    begin
                      d6[j]:= 0.0;
                    end;
                    d6[col]:= sigma;
                    sum := 0.0;
                    for j:= col to nr6 do
                    begin
                       sum := sum + sqr(m16[j,col]-d6[j]);
                    end;
                    for j:= col to nr6 do
                    begin
                        for k:= col to nr6 do
                        begin
                             begin
                                if abs(sum) < 0.00001
                               then  temp:= 0.0
                              else temp := -2*(m16[j,col]-d6[j])*(m16[k,col]-
d6[k])/sum;

                             end;
                                if j = k then
                                begin
                                   temp:= 1.0 + temp;
                                end;
                          h6[j,k]:= temp;
                        end;
                    end;
                   for j:= col to nr6 do
                   begin
                        for k:= col to nc6 do
                        begin
                             temp:= 0.0;
                             for l:= col to nr6 do
                             begin
                                  temp:= temp + h6[j,l]*m16[l,k];
                             end;
```

```
                                   c6[j,k]:= temp;
                           end;
                        end;
                       { m16:= c6;}


                       for i:=col to nr6 do
                       begin
                       for j:=col to nc6 do
                       begin
                       m16[i,j]:=c6[i,j];
                       end;
                       end;
                  end;
                        for i:= 2 to nr6 do
                        begin
                             for j:= 1 to i-1 do
                             begin
                                  m16[i,j]:= 0.0;
                             end;
                        end;
              end;

             procedure kgain(var nx5: integer;var spt5,f5: trytry;
                           var  g5,h5:try;var sr5,sq5:real;var khat5: try);
                  var
                     ny, i,j,nu,k,nytot,nx5nu: integer;
                      m15: try40;
                      sptt : trytry;
                       begin
                              m15[1,1]:= sr5;
                              ny:= 1 ;
                              nu:= 1;
                              for i:= 1 to nx5 do
                              begin
                                   for j:= 1 to nx5 do
                                   begin
                                        m15[ny+i,ny+j]:= spt5[j,i];
                                   end;
                              end;
                              for i:= 1 to nx5 do
                              begin
                                   m15[ny+i,1]:= 0.0;
                                   for k:= 1 to nx5 do
                                   begin
                                        m15[ny+i,1]:= m15[ny+i,1]+spt5[k,i]*h5[k];
                                   end;
                              end;
                              for j:= ny+1 to ny+nx5 do
                              begin
                                   m15[1,j]:= 0.0;
                              end;
                              nytot:= ny+nx5;
                            {  writeln ('enter hholder first time'); }
              {  writeln('m15',m15[1,1],m15[1,2],m15[1,3]); }
                hholder(nytot,nytot,m15);
               { writeln('m15after hholder frst ',m15[1,1],m15[1,2],m15[1,3]);}
                {*******************************************}
if abs(m15[1,1]) <= 0.0000001 then writeln('R is singular');
                for i:= 1 to nx5 do
                begin
                     khat5[i]:= m15[1,i+1]/m15[1,1];
             {            writeln('khat5',khat5[i],'    ',i);}
                     for j:= 1 to nx5 do
                     begin
                          sptt[j,i]:= m15[ny+i,ny+j];
                     end;
                end;
                for i:= 1 to nx5 do
                begin
                     for j:= 1 to nx5 do
                     begin
                          m15[i,j]:= 0.0;
                          for k:= 1 to nx5 do
                          begin
                               m15[i,j]:= m15[i,j]+sptt[k,i]*f5[j,k];
                          end;
                     end;
                end;
                for j:= 1 to nx5 do
                begin

                     m15[nx5+1,j]:= sq5*g5[j];
                end;
                nx5nu := nx5+nu;
              {  writeln('enter hholder second time'); }
                hholder(nx5nu,nx5,m15);
                for i:= 1 to nx5 do
                begin
                     for j:= 1 to nx5 do
                     begin
                          spt5[j,i]:= m15[i,j];
                     end;
                end;
           end;


 begin              {******beginning of estimater******}
          {Writeln('time=',time,'initial=',initial);}
           if initial = 1
             then
                 begin
                      initializeArrays;
                      goto 3;
                 end;
           ud:=uf;
           filter(up,uf,xu,fc,n5);
           yd := yf;
```

```
             filter(yg,yf,xy,fc,nf);
             zdd := zd;
             zd := zf;
             filter(z,zf,xz,fc,nf);
             update(phi,ud,yd,ns);
             phi[nx]:=1.0;
             nsnf := ns + nf;
             time:=time+1;
             if time <= nsnf then goto 3;

          { ***** estimate bias ***** }

             kgain(nx,spt,f,g,phi,sr,sq,khat);
            { write('phi and yf into newest',phi[1],phi[2],phi[3],yf);}
            { write('khat',khat[1],khat[2],khat[3]);                    }
             newest(nx,xhat,khat,phi,yf);
               { writeln('paras=',
                   xhat[1],xhat[2],xhat[3],xhat[4],xhat[5]) ;}

             {*** compute bias estimate ***}

             i:= ns;
             while i >= 2 do
             begin
                  beta[i]:= beta[i - 1];
                  i:= i - 1 ;
             end;
             beta[1]:= bias;
             bias:= xhat[nx];
{             Writeln('bias=',bias);}
for j:= 1 to ns do
             begin
                  bias:= bias + xhat[j]*beta[j];
             end;
{            a1:=xhat[1];a2:=xhat[2];a3:=xhat[3];}


{********* estimate *********
             phi2[1] := yf;
             phi2[2] := 1.0;
             zf2 := (zf - 2.0*zd + zdd)*sqr(fs);
             n2 := 2;
             kgain(n2,spt2,f,g2,phi2,sr2,sq2,khat2);

             newest(n2,xhat2,khat2,phi2,zf2);

             if abs(xhat2[1]) > 0.00000001
             then gbias:= xhat2[2]/xhat2[1];
             vhat:= xhat2[1];

             *** estimate pitch angle ***

             f3[1,1]:= 1.0;
             f3[1,2]:= -1/fs;
```

```
              f3[2,1]:= 0.0;
              f3[2,2]:= 1.0;
              g3[1]:= 1.0;
              g3[2]:= 1.0;
              h3[1]:= 1.0;
              h3[2]:= 0.0;
              kgain(n2,spt3,f3,g3,h3,sr3,sq3,khat3);  }

             {observer for plytch ajh,10/2/88}
             begin
             for i:=1 to 3 do
             begin
             xnew[i]:=0.0;
             for j:=1 to 3 do
             xnew[i]:=xnew[i]+f3[i,j]*xs[j]+g3[i]*up/10.0+khat3[i]*(z-xs[3]+yq-xs[1
]);
             end;
             end;pitch:=xnew[2];
             xs[1]:=xnew[1];
             xs[2]:=xnew[2];
             xs[3]:=xnew[3];
             a1:=xs[1];a2:=xs[2];a3:=xs[3];


 3:end;



procedure GenerateDiveplaneCommand (var  auvdepth,auvpitch,auvpitchrate,
                auvdepthcom,delta: real;var k:integer );

{ This procedure takes digitized voltage values of depth,pitch,pitchrate,
  and target depth in volys and sends new commands for control.}

const

{ DT 2801-A DIGITAL TO ANALOG Conversion declarations                    }

   d2achannel0 = 0;
   pfs = 10.0;
   mfs = - 10.0;
   noc = 4096;
   T=0.2;amp=0.05;T0=2.0;          {these parameters include sin excitation}
  { g1=0.3066;g2=0.8604;g3=-0.2493;}
  {these gains are for 20 Hz.@ poles=[.92 .925 .926]}
{   g1=-0.2286;g2=0.1944;g3=-0.0118;
  these gains are for 5Hz. and poles & [0.92 .925 .926]}
  g1= 0.2060;g2=0.6573;g3=-0.1605;{for5Hz. at [0..85 0.860 0.870]}

var
   digitaldatavalue0, status          : integer;
   e1,e2,e3                           : real;

function ConvertAnalog2Digital ( analogvalue : real ): integer;
```

```
{ This function converts analog signal volts to an equivalent
  digital value.  See App D of PCLAB book.                   }

var
   temp              : real;
begin

   temp :=  ( analogvalue - mfs ) * ( (noc - 10) / (pfs - mfs ) );
   convertanalog2digital := round ( temp );

end;

begin  { ------------- GenerateDivePlaneCommand ------------------- }
       e3:=(auvdepthcom-auvdepth);
       e2:=(auvpitch);
       e1:=(auvpitchrate);

       if   abs(e3)<= 0.000001   then e3:=0.0;
       if abs(e1)<=0.000005 then  e1:=0.0;
       if abs(e2) <=0.000005 then  e2:=0.0;

       delta   :=g3*e3 + g1*(-e1) +g2*(-e2);

     if(abs(delta)>0.1) then
       begin
        delta :=0.1*abs(delta)/delta;
       end;
   delta := 10*delta;
   digitaldatavalue0 := convertanalog2digital ( delta   );
   status := dacvalue ( d2achannel0, digitaldatavalue0 );
{   status := terminate;   }

end;   { ------------- GenerateDivePlaneCommand ------------------- }

procedure Move1(var  r0,u,r :real);
const
countmax=400;
count2=200;
{type
activecontrolmode = (run,reset,exit);
auvattitude = (climb,maintain,diving); }


var
   count   :integer   ;
   dr,x1,x2,x3   :real ;
   activemode :activeCONTROLmode;
   attitude :auvattitude;

begin
       activemode:= reset;
       attitude:=climb;
       count:=0 ;
```

```
    while(count<countmax)   do
       begin
       dr:=0.0'0;
       GetDigitalSensoryData(x1,x2,x3);
       if(count>count2)then dr:=0.0;
       r:=r0+dr*count;
       {if(count>count2)then dr:=0.0;}
       GenerateDiveplaneCommand (x1,x2,x3,r,u,count);
       count:=count+1;
     {  UpdateRunModeScreen(x3,x2,x1,r,activemode,attitude)  }
       end;
end;


procedure InitializeParameters ;

{ This procedure initializes all declared control and display parameters to
  zero.                                                                      }

begin                              { ---- procedure InitializeParameters ---   }
     auvpitchvolts :=0.0 ;
     auvdepthvolts := 0.0;
     auvspeedvolts := 0.0;
     auvpitchratevolts := 0.0;
     auvdepth := 0.0;
     auvspeed := 0.0;
     auvpitchrate := 0.0;
     estdepth:=0.0;
     err:=0.0;
end;                               { ---- procedure InitializeParameters ---   }


begin              { --------------------- ActiveControl ----------------- }


initializeparameters;
initial := 1;      {initializeArrays in EST procedure}
5:
time := 1;         {initializes beginning of data file}
clrscr;
writeln('DATA FILE NAME? [ie f082201.dat]');
readln(filename);
assign(filevar,filename);
rewrite(filevar);
clrscr;
activemode := run;

repeat          { -------------- Repeat until activemode = exit  ------ }


    { ClosedLoopControlScreen;                     }
      GetTargetDepth ( targetdepth, targetdepthvolts );
      RunModeScreen;
```

```
        1:
        while ( not  keyhit ( modereply, modereply2))   do
           begin
              updatecounter := 0;

              while ( updatecounter <  updateincrement ) do
                begin
        GetDigitalSensoryData (auvdepthvolts,auvpitchvoltsxx,auvpitchratevolts);


      GenerateDiveplaneCommand ( ac3,ac2,ac1,
                          targetdepthvolts,divevolts,time );
       EST (divevolts,auvpitchratevolts,auvdepthvolts,
            speed,auvpitchvolts,initial,bias1,ac1,ac2,ac3);
       writeln(filevar,time:5,auvdepthvolts:12:6,auvpitchvoltsxx:12:6,
               auvpitchratevolts:12:6,auvpitchvolts:8:3,divevolts:12:6,
               bias1:12:6,ac1:12:6,ac2:12:6,ac3:12:6);


         ATTITUDE_ (targetdepthvolts,auvdepthvolts,attitude);
                 initial := 0;
                updatecounter := updatecounter + 1;
                time := time + 1;
              end;   { while updatecounter < updateincrement  }


      UpdateRunModeScreen ( auvdepthvolts, auvpitchvoltsxx,
         auvpitchratevolts,divevolts,targetdepth, activemode, attitude,bias1);

       end;         { while not KeyHit                          }

        if (ord(modereply) = 27) and (ord(modereply2) = 59) then  { KeyHit=F1 }
        begin
        close(filevar);
        goto 5;
        end
        else if (ord(modereply) = 27) and (ord(modereply2) = 60) then
                begin

                   activemode := reset;
                   Move1(targetdepthvolts,divevolts,tgtnew);
                   targetdepthvolts:=tgtnew  ;
                   GOTO 1
                   end

        else if (ord(modereply) = 27) and (ord(modereply2) =61) then
        begin
             close(filevar);
          activemode := exit;
        end                                       { KeyHit= f3}
     until   (activemode=exit)

  end;         { ----------------------- ActiveControl   ------------------- }
```

```
begin         { ----------------------- StatusAndCommand ------------------ }

   repeat
      StatusAndCommandScreen;
      GetKey ('',mode,mode2);
      if ( ord (mode) = 27 ) and ( ord (mode2) = 59 ) then
         begin
              clrbox2 (x1,y1,x2,y2);
              ClosedLoopControl;
         end;
   until ( mode in ['Q','q'] );

end;          { ----------------------- StatusAndCommand ------------------ }


procedure InitializeZeroDigitalSignalOut;

{ This procedure MUST be executed as the first procedure called in the main
program to insure a zero signal out on the 2 output channels.  Otherwise the
DT 2801-A board defaults to a minimum full scale output.                   }

const


   digitalchan0        = 0;
   digitalchan1        = 1;
   digitalcommandboard = 1;

var
   status,
   digitaldatavalue    : integer;


begin
   digitaldatavalue := 2048;      { This will be converted to an equivalent
                                    zero analog signal out on a 12 bit
                                    resolution converter like DT 2801-A.     }

   status := initialize;
   status := selectboard ( digitalcommandboard );
   status := dacvalue ( digitalchan0, digitaldatavalue );
   status := dacvalue ( digitalchan1, digitaldatavalue );
   status := terminate;
end;

procedure DeactivateADBoardAndExitProgram;

{ This procedure deactivates the DT 2801-A board and presents an exit screen.}

var
```

```
status              : integer;

begin  { ------------------- DeactivateADBoardAndExitProgram ---------------- }
  status := terminate;
  clrbox2 (x1,y1,x2,y2);
  boxprint (y1+10,x1,x2,'THIS CONCLUDES YOUR AUV AUTOPILOTTING SESSION , BYE');

end;    { ------------------- DeactivateADBoardAndExitProgram ---------------- }



BEGIN         { -------------------------------     MAIN PROGRAM   ---------- }

   InitializeZeroDigitalSignalOut;
   clrscr;
   repeat

      MainMenu ( option );

     if ( option in ['R','r']) then
         begin
            repeat
              begin
                 StatusAndCommand ( controlmode );
              end;
            until ( controlmode in ['q','Q']);
         end;
    until ( option in ['Q','q']);

    DeactivateADBoardAndExitProgram;

END.          { -------------------------------- MAIN PROGRAM    ----------  }
```

LIST OF REFERENCES

1. 5th International Symposium on Unmanned Untethered Submersible Technology, University of New Hampshire, June 22-24, 1987, Symposium Proceedings.

2. Healey, A.J., Cristi, R., Smith, D.L., McGhee, R.B., "Navigation, Path Planning, Dynamics and Control of Generic Autonomous Underwater Vehicles Proposal," Research proposal to Naval Postgraduate School Direct Research Fund, Monterey, California, April 1988.

3. Boncal, R.J., A Study of Model Based Maneuvering Controls for Autonomous Underwater Vehicles, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1987.

4. Brunner, G.M., Experimental Verification of AUV Performance, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.

5. DT/Notebook, Laboratory Technologies Corporation, Wilmington, Massachusetts, 1986.

6. MATRIXx, Integrated System Inc., Santa Clara, California, 1987.

7. Delaplane, S.W., Preliminary Design and Cycle Verification of a Digital Autopilot for Autonomous Underwater Vehicles, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1988.

8. D'azzo, J.J. and Houpis, C.H., Linear Control System Analysis and Design Conventional and Modern, 3rd Ed., McGraw-Hill, 1988.

9. Turbo Pascal Tutor, Borland International, Scotts Valley, California, 1987.

10. Turbo Pascal 4.0, Borland International, Scotts Valley, California, 1987.

11. Swan, T., Mastering Turbo Pascal, Hayden Books, Indianapolis, Indiana, 1987

12. Johnstone, R.S., Fries, D.W., "Simulation of a Submerged Autonomous Vehicle with Inertial Navigation," Paper, Advanced Marine Systems--Lockheed Missiles and Space Company, Inc., Sunnyvale, California, presented at DARPA/CSDL Symposium, June 1988.

13. Cannon, R.H. Jr., Dynamics of Physical Systems, McGraw-Hill, New York, 1967.

14. User Manual for DT 2801 Series, Single Board, Analog and Digital I/O Systems for the IBM PC, Data Translation, Inc., Marlborough, Massachusetts, 1986.

15. User Manual for PCLAB, Sp041 v2.00, Data Translation, Inc., Marlborough, Massachusetts, 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chairman, Code 69Hy Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	5
4.	Professor D.L. Smith, Code 69Sm Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93943-5000	1
5.	Professor R. McGhee, Code 52Mz Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	1
6.	Professor R. Christi, Code 62Cx Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
7.	Dr. G. Dobeck, Code 4210 NCSC Panama City, Florida 32407-5000	1
8.	Russ Werneth, Code U25 Naval Surface Weapons Center White Oak, Maryland 20910	1
9.	Paul Heckman, Code 943 Head, Undersea AI & Robotics Branch Naval Ocean System Center San Diego, California 92152	1
10.	Dr. D. Milne, Code 1563 DTRC, Carderock Bethesda, Maryland 20084-5000	1

11.	Naval Sea Systems Command Code PMS-350, Attn: Ms. Judy Rumsey Washington, D.C. 20361-5101	1
12.	LT Relle L. Lyman, Jr. Navsea Robotics Office Naval Sea Systems Command Washington, D.C. 20362-5101	1
13.	Distinguished Professor G. Thaler, Code 62Tr Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, California 93943-5000	1
14.	LT Gerard J. Reina, USN 107 Seafoam Avenue Monterey, California 93940	1
15.	CDR Gordon Mac Donald 1 Shubrick Road Monterey, California 93940	1